Featherweight Java: A Minimal Core Calculus for Java and GJ

ATSUSHI IGARASHI
University of Tokyo
BENJAMIN C. PIERCE
University of Pennsylvania
and
PHILIP WADLER
Avaya Labs

Several recent studies have introduced lightweight versions of Java: reduced languages in which complex features like threads and reflection are dropped to enable rigorous arguments about key properties such as type safety. We carry this process a step further, omitting almost all features of the full language (including interfaces and even assignment) to obtain a small calculus, Featherweight Java, for which rigorous proofs are not only possible but easy. Featherweight Java bears a similar relation to Java as the lambda-calculus does to languages such as ML and Haskell. It offers a similar computational "feel," providing classes, methods, fields, inheritance, and dynamic typecasts with a semantics closely following Java's. A proof of type safety for Featherweight Java thus illustrates many of the interesting features of a safety proof for the full language, while remaining pleasingly compact. The minimal syntax, typing rules, and operational semantics of Featherweight Java make it a handy tool for studying the consequences of extensions and variations. As an illustration of its utility in this regard, we extend Featherweight Java with *generic classes* in the style of GJ (Bracha, Odersky, Stoutamire, and Wadler) and give a detailed proof of type safety. The extended system formalizes for the first time some of the key features of GJ.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory; D.3.2 [**Programming Languages**]: Language Classifications—*Object-oriented languages*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Classes and objects*;

This is a revised and extended version of a paper presented in the Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'99), ACM SIGPLAN Notices volume 34 number 10, pages 132–146, October 1999. This work was done while Igarashi was visting the University of Pennsylvania as a research fellow of the Japan Society of the Promotion of Science. Pierce was supported by the University of Pennsylvania and the National Science Foundation under grant CCR-9701826, *Principled Foundations for Programming with Objects*.

Authors' addresses: A. Igarashi, Department of Graphics and Computer Science, Graduate School of Arts and Sciences, University of Tokyo, 3-8-1 Komaba, Meguro-ku, Tokyo 153-8902, Japan; email: igarashi@graco.c.u-tokyo.ac.jp; B. C. Pierce, Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd Street, Philadelphia, PA 19104-6389; email: bcpierce@cis.upenn.edu; P. Wadler, 233 Mount Airy Road, Basking Ridge, NJ 07920; email: wadler@avaya.com.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2001 ACM 0098-3500/01/0500–0396 $5.00

Polymorphism; F.3.3 [**Logics and Meaning of Programs**]: Studies of Program Constructs—*Object-oriented constructs*

General Terms: Design, Languages, Theory

Additional Key Words and Phrases: Compilation, generic classes, Java, language design, language semantics

1. INTRODUCTION

"Inside every large language is a small language struggling to get out..."
T. Hoare[1]

Formal modeling can offer a significant boost to the design of complex real-world artifacts such as programming languages. A formal model may be used to describe some aspect of a design precisely, to state and prove its properties, and to direct attention to issues that might otherwise be overlooked. In formulating a model, however, there is a tension between completeness and compactness: The more aspects the model addresses at the same time, the more unwieldy it becomes. Often it is sensible to choose a model that is less complete but more compact, offering maximum insight for minimum investment. This strategy may be seen in a flurry of recent papers on the formal properties of Java, which omit advanced features such as concurrency and reflection and concentrate on fragments of the full language to which well-understood theory can be applied.

We propose Featherweight Java, or FJ, as a new contender for a *minimal* core calculus for modeling Java's type system. The design of FJ favors compactness over completeness almost obsessively, having just five forms of expression: object creation, method invocation, field access, casting, and variables. Its syntax, typing rules, and operational semantics fit comfortably on a few pages. Indeed, our aim has been to omit as many features as possible—even assignment—while retaining the core features of Java typing. There is a direct correspondence between FJ and a purely functional core of Java, in the sense that every FJ program is literally an executable Java program.

FJ is only a little larger than Church's lambda calculus [Barendregt 1984] or Abadi and Cardelli's object calculus [1996], and is significantly smaller than previous formal models of class-based languages like Java, including those put forth by Drossopoulou et al. [1999], Syme [1997], Nipkow and von Oheimb [1998], and Flatt et al. [1998a; 1998b]. Being smaller, FJ lets us focus on just a few key issues. For example, we have discovered that

[1]We thank Tony Hoare, to whom the first quote below is attributed, for informing us of the second one:

Inside every large program is a small program struggling to get out...
— T. Hoare, Efficient Production of Large Programs (1970)

I'm fat, but I'm thin inside.
Has it ever struck you that there's a thin man inside every fat man?
—George Orwell, *Coming Up For Air* (1939)

capturing the behavior of Java's cast construct in a traditional "small-step" operational semantics is trickier than we would have expected, a point that has been overlooked or underemphasized in other models.

One use of FJ is as a starting point for modeling languages that extend Java. Because FJ is so compact, we can focus attention on essential aspects of the extension. Moreover, because the proof of soundness for pure FJ is very simple, a rigorous soundness proof for even a significant extension may remain manageable. The second part of the article illustrates this utility by enriching FJ with generic classes and methods *à la* GJ [Bracha et al. 1998]. The model omits some important aspects of GJ (such as "raw types" and type argument inference for generic method calls). Nonetheless, it led to the discovery and repair of one bug in the GJ compiler and, more importantly, has been a useful tool in clarifying our thought. Because the model is small, it is easy to contemplate further extensions, and we have begun the work of adding raw types to the model; so far, this has revealed at least one corner of the design that was underspecified.

Our main goal in designing FJ was to make a proof of type soundness ("well-typed programs do not get stuck") as concise as possible, while still capturing the essence of the soundness argument for the full Java language. Any language feature that made the soundness proof *longer* without making it significantly *different* was a candidate for omission; we also dropped features that did not appear to interact with polymorphism in significant ways. As in previous studies of type soundness in Java, we do not treat advanced mechanisms such as concurrency, inner classes, and reflection. In addition, the Java features omitted from FJ include assignment, interfaces, overloading, messages to `super`, `null` pointers, base types (`int`, `bool`, etc.), abstract method declarations, shadowing of superclass fields by subclass fields, access control (`public`, `private`, etc.), and exceptions. The features of Java that we *do* model include mutually recursive class definitions, object creation, field access, method invocation, method override, method recursion through `this`, subtyping, and casting.

One key simplification in FJ is the omission of assignment. In essence, all fields and method parameters in FJ are implicitly marked `final`: we assume that an object's fields are initialized by its constructor and never changed afterward. This restricts FJ to a "functional" fragment of Java, in which many common Java idioms, such as use of enumerations, cannot be represented. Nonetheless, this fragment is computationally complete (it is easy to encode the lambda calculus into it), and is large enough to include many useful programs (many of the programs in Felleisen and Friedman's Java text [1998] use a purely functional style). Moreover, most of the tricky typing issues in both Java and GJ are independent of assignment. An important exception is that the type inference algorithm for generic method invocation in GJ has some twists imposed on it by the need to maintain soundness in the presence of assignment. This article treats a simplified version of GJ without type inference.

The remainder of this article is organized as follows. Section 2 introduces the main ideas of Featherweight Java, presents its syntax, type rules, and reduction rules, and develops a type soundness proof. Section 3 extends

Featherweight Java to Featherweight GJ, which includes generic classes and methods. Section 4 presents an erasure map from FGJ to FJ, modeling the techniques used to compile GJ into Java. Section 5 discusses related work, and Section 6 concludes.

2. FEATHERWEIGHT JAVA

In FJ, a program consists of a collection of class definitions plus an expression to be evaluated. (This expression corresponds to the body of the `main` method in full Java.) Here are some typical class definitions in FJ.

```
class A extends Object {
  A() { super(); }
}
class B extends Object {
  B() { super(); }
}
class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst=fst; this.snd=snd;
  }
  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd);
  }
}
```

For the sake of syntactic regularity, we always (1) include the supertype (even when it is `Object`); (2) write out the constructor (even for the trivial classes `A` and `B`); and (3) write the receiver for a field access (as in `this.snd`) or a method invocation, even when the receiver is `this`. Constructors always take the same stylized form: there is one parameter for each field, with the same name as the field; the `super` constructor is invoked on the fields of the supertype; and the remaining fields are initialized to the corresponding parameters. In this example the supertype is always `Object`, which has no fields, so the invocations of `super` have no arguments. Constructors are the only place where `super` or `=` appears in an FJ program. Since FJ provides no side-effecting operations, a method body always consists of `return` followed by an expression, as in the body of `setfst()`.

In the context of the above definitions, the expression

```
new Pair(new A(), new B()).setfst(new B())
```

evaluates to the expression

```
new Pair(new B(), new B()).
```

There are five forms of expression in FJ. Here, `new A()`, `new B()`, and `new Pair(e1, e2)` are *object constructors*, and `e3.setfst(e4)` is a *method* *invocation*. In the body of `setfst`, the expression `this.snd` is a *field access*, and the occurrences of `newfst` and `this` are *variables*. (The syntax of FJ differs from Java in that `this` is a variable rather than a keyword). The remaining form of expression is a *cast*. The expression

```
((Pair)new Pair(new Pair(new A(), new B()), new A()).fst).snd
```

evaluates to the expression

```
new B().
```

Here, `((Pair)e5)`, where `e5` is `new Pair(...).fst`, is a cast. The cast is required because `e5` is a field access to `fst`, which is declared to contain an `Object`, whereas the next field access, to `snd`, is only valid on a `Pair`. At run time, it is checked whether the `Object` stored in the `fst` field is a `Pair` (and in this case the check succeeds).

In Java, we may prefix a field or parameter declaration with the keyword `final` to indicate that it may not be assigned to, and all parameters accessed from an inner class must be declared `final`. Since FJ contains no assignment and no inner classes, it matters little whether or not `final` appears, so we omit it for brevity.

Dropping side effects has a pleasant side effect: evaluation can be easily formalized entirely within the syntax of FJ, with no additional mechanisms for modeling the heap. Moreover, in the absence of side effects, the order in which expressions are evaluated does not affect the final outcome (modulo nontermination), so we can define the operational semantics of FJ straightforwardly using a nondeterministic small-step reduction relation, following long-standing tradition in the lambda calculus. Of course, Java's call-by-value evaluation strategy is subsumed by this more general relation, so the soundness properties we prove for reduction will hold for Java's evaluation strategy as a special case.

There are three basic computation rules: one for field access, one for method invocation, and one for casts. Recall that, in the lambda calculus, the beta-reduction rule for applications assumes that the function is first simplified to a lambda abstraction. Similarly, in FJ the reduction rules assume the object operated upon is first simplified to a `new` expression. Thus, just as the slogan for the lambda calculus is "everything is a function," here the slogan is "everything is an object."

The following example shows the rule for field access in action:

$$\texttt{new Pair(new A(), new B()).snd} \rightarrow \texttt{new B()}$$

Due to the stylized form for object constructors, we know that the constructor has one parameter for each field, in the same order that the fields are declared. Here the fields are `fst` and `snd`, and an access to the `snd` field selects the second parameter.

Here is the rule for method invocation in action (/ denotes substitution):

$$\begin{array}{l}\texttt{new Pair(new A(), new B()).setfst(new B())}\\ \rightarrow \left[\begin{array}{l}\texttt{new B()/newfst,}\\ \texttt{new Pair(new A(),new B())/this}\end{array}\right] \texttt{new Pair(newfst, this.snd)}\\ \text{i.e., } \texttt{new Pair(new B(), new Pair(new A(), new B()).snd)}\end{array}$$

The receiver of the invocation is the object `new Pair(new A(), new B())`, so we look up the `setfst` method in the `Pair` class, where we find that it has formal parameter `newfst` and body `new Pair(newfst, this.snd)`. The invocation reduces to the body with the formal parameter replaced by the actual, and the special variable `this` replaced by the receiver object. This is similar to the beta rule of the lambda calculus, $(\lambda\texttt{x.e0})\texttt{e1} \to [\texttt{e1/x}\ ]\texttt{e0}$. The key differences are the fact that the class of the receiver determines where to look for the body (supporting method override), and the substitution of the receiver for `this` (supporting "recursion through self"). Readers familiar with Abadi and Cardelli's Object Calculus will see a strong similarity to their ς reduction rule [Abadi and Cardelli 1996]. In FJ, as in the lambda calculus and the pure Abadi-Cardelli calculus, if a formal parameter appears more than once in the body it may lead to duplication of the actual, but since there are no side effects this causes no problems.

Here is the rule for a cast in action:

$$\texttt{(Pair)new Pair(new A(), new B())} \to \texttt{new Pair(new A(), new B())}$$

Once the subject of the cast is reduced to an object, it is easy to check that the class of the constructor is a subclass of the target of the cast. If so, as is the case here, then the reduction removes the cast. If not, as in the expression `(A)new B()`, then no rule applies and the computation is *stuck*, denoting a runtime error.

There are three ways in which a computation may get stuck: an attempt to access a field not declared for the class; an attempt to invoke a method not declared for the class ("message not understood"); or an attempt to cast to something other than a superclass of an object's runtime class. We prove that the first two of these never happen in well-typed programs, and the third never happens in well-typed programs that contain no downcasts (and no "stupid casts"—a technicality explained below).

As usual, we allow reductions to apply to any subexpression of an expression. Here is a computation for the second example expression above, where the next subexpression to be reduced is underlined at each step.

$$\begin{array}{ll}
 & \texttt{((Pair)}\underline{\texttt{new Pair(new Pair(new A(), new B()), new A()).fst}}\texttt{).snd} \\
\to & \underline{\texttt{((Pair)new Pair(new A(),new B()))}}\texttt{.snd} \\
\to & \underline{\texttt{new Pair(new A(), new B()).snd}} \\
\to & \texttt{new B()}
\end{array}$$

We prove a type soundness result for FJ: if a well-typed expression `e` reduces to a normal form, an expression that cannot reduce any further, then the normal form is either a well-typed value (an expression consisting only of `new`), whose type is a subtype of the type of `e`, or stuck at a failing typecast.

With this informal introduction in mind, we may now proceed to a formal definition of FJ.

2.1 Syntax

The abstract syntax of FJ class declarations, constructor declarations, method declarations, and expressions is given at the top of Figure 1. The metavariables `A`, `B`, `C`, `D`, and `E` range over class names; `f` and `g` range over field names; `m` ranges

Syntax:

$\texttt{L} ::= \texttt{class C extends C \{}\bar{\texttt{C}}\ \bar{\texttt{f}}\texttt{; K }\bar{\texttt{M}}\texttt{\}}$

$\texttt{K} ::= \texttt{C(}\bar{\texttt{C}}\ \bar{\texttt{f}}\texttt{)\{super(}\bar{\texttt{f}}\texttt{); this.}\bar{\texttt{f}}\texttt{=}\bar{\texttt{f}}\texttt{;\}}$

$\texttt{M} ::= \texttt{C m(}\bar{\texttt{C}}\ \bar{\texttt{x}}\texttt{)\{ return e; \}}$

$\texttt{e} ::= \texttt{x} \mid \texttt{e.f} \mid \texttt{e.m(}\bar{\texttt{e}}\texttt{)} \mid \texttt{new C(}\bar{\texttt{e}}\texttt{)} \mid \texttt{(C)e}$

Subtyping:

$$\texttt{C <: C} \qquad\qquad \frac{\texttt{C <: D} \quad \texttt{D <: E}}{\texttt{C <: E}} \qquad\qquad \frac{\texttt{class C extends D \{...\}}}{\texttt{C <: D}}$$

Field lookup:

$$\mathit{fields}(\texttt{Object}) = \bullet$$

$$\frac{\texttt{class C extends D \{}\bar{\texttt{C}}\ \bar{\texttt{f}}\texttt{; K }\bar{\texttt{M}}\texttt{\}} \qquad \mathit{fields}(\texttt{D}) = \bar{\texttt{D}}\ \bar{\texttt{g}}}{\mathit{fields}(\texttt{C}) = \bar{\texttt{D}}\ \bar{\texttt{g}}, \bar{\texttt{C}}\ \bar{\texttt{f}}}$$

Method type lookup:

$$\frac{\texttt{class C extends D \{}\bar{\texttt{C}}\ \bar{\texttt{f}}\texttt{; K }\bar{\texttt{M}}\texttt{\}} \qquad \texttt{B m(}\bar{\texttt{B}}\ \bar{\texttt{x}}\texttt{)\{ return e; \}} \in \bar{\texttt{M}}}{\mathit{mtype}(\texttt{m}, \texttt{C}) = \bar{\texttt{B}} \rightarrow \texttt{B}}$$

$$\frac{\texttt{class C extends D \{}\bar{\texttt{C}}\ \bar{\texttt{f}}\texttt{; K }\bar{\texttt{M}}\texttt{\}} \qquad \texttt{m} \not\in \bar{\texttt{M}}}{\mathit{mtype}(\texttt{m}, \texttt{C}) = \mathit{mtype}(\texttt{m}, \texttt{D})}$$

Method body lookup:

$$\frac{\texttt{class C extends D \{}\bar{\texttt{C}}\ \bar{\texttt{f}}\texttt{; K }\bar{\texttt{M}}\texttt{\}} \qquad \texttt{B m(}\bar{\texttt{B}}\ \bar{\texttt{x}}\texttt{)\{ return e; \}} \in \bar{\texttt{M}}}{\mathit{mbody}(\texttt{m}, \texttt{C}) = \bar{\texttt{x}}\texttt{.e}}$$

$$\frac{\texttt{class C extends D \{}\bar{\texttt{C}}\ \bar{\texttt{f}}\texttt{; K }\bar{\texttt{M}}\texttt{\}} \qquad \texttt{m} \not\in \bar{\texttt{M}}}{\mathit{mbody}(\texttt{m}, \texttt{C}) = \mathit{mbody}(\texttt{m}, \texttt{D})}$$

Fig. 1. FJ: Syntax, subtyping rules, and auxiliary functions.

over method names; `x` ranges over variables; `d` and `e` range over expressions; `L` ranges over class declarations; `K` ranges over constructor declarations; and `M` ranges over method declarations. We assume that the set of variables includes the special variable `this`, which cannot be used as the name of an argument to a method. (As we will see later, the restriction is imposed by the typing rules). Instead, it is considered to be implicitly bound in every method declaration. The evaluation rule for method invocation will have the job of substituting an appropriate object for `this`, in addition to substituting the argument values for the parameters. Note that since we treat `this` in method bodies as an ordinary variable, no special syntax for it is required.

We write $\bar{\texttt{f}}$ as shorthand for a possibly empty sequence $\texttt{f}_1, \ldots, \texttt{f}_n$ (and similarly for $\bar{\texttt{C}}$, $\bar{\texttt{x}}$, $\bar{\texttt{e}}$, etc.) and write $\bar{\texttt{M}}$ as shorthand for $\texttt{M}_1 \ldots \texttt{M}_n$ (with no commas). We write the empty sequence as $\bullet$ and denote concatenation of sequences using a comma. The length of a sequence $\bar{x}$ is written $\#(\bar{x})$. We abbreviate operations on pairs of sequences in the obvious way, writing "$\bar{C}$ $\bar{f}$" for "C_1 f_1,...,C_n f_n", where n is the length of $\bar{C}$ and $\bar{f}$, and similarly "$\bar{C}$ $\bar{f}$;" as shorthand for the sequence of declarations "C_1 f_1;...C_n f_n;" and "this.$\bar{f}$=$\bar{f}$;" as shorthand for "this.f_1=f_1;...;this.f_n=f_n;". Sequences of field declarations, parameter names, and method declarations are assumed to contain no duplicate names. As in Java, we assume that casts bind less tightly than other forms of expression.

The class declaration `class C extends D {`$\bar{C}$ $\bar{f}$; K $\bar{M}$`}` introduces a class named `C` with superclass `D`. The new class has fields $\bar{f}$ with types $\bar{C}$, a single constructor `K`, and a suite of methods $\bar{M}$. The instance variables declared by `C` are added to the ones declared by `D` and its superclasses, and should have names distinct from these. (In full Java, instance variables of superclasses may be redeclared, in which case the redeclaration shadows the original in the current class and its subclasses. We omit this feature in FJ). The methods of `C`, on the other hand, may either override methods with the same names that are already present in `D` or add new functionality special to `C`.

The constructor declaration C($\bar{D}$ $\bar{g}$; $\bar{C}$ $\bar{f}$){super($\bar{g}$); this.$\bar{f}$=$\bar{f}$;} shows how to initialize the fields of an instance of `C`. Its form is completely determined by the instance variable declarations of `C` and its superclasses: it *must* take exactly as many parameters as there are instance variables, and its body *must* consist of a call to the superclass constructor to initialize its fields from the parameters $\bar{g}$, followed by an assignment of the parameters $\bar{f}$ to the new fields of the same names declared by `C`. (These constraints are actually enforced by the typing rule for classes in Figure 2).

The method declaration D m($\bar{C}$ $\bar{x}$){ return e; } introduces a method named `m` with result type `D` and parameters $\bar{x}$ of types $\bar{C}$. The body of the method is the single statement `return e;`. The variables $\bar{x}$ and the special variable `this` are bound in `e`. As we will see later, the typing rules prohibit `this` from appearing as a method parameter name.

A class table CT is a mapping from class names `C` to class declarations `L`. A program is a pair (CT,e) of a class table and an expression. To lighten the notation in what follows, we always assume a *fixed* class table CT.

Every class has a superclass, declared with `extends`. This raises a question: What is the superclass of the class `Object`? There are various ways to deal with this issue; the simplest one that we have found is to take `Object` as a distinguished class name whose definition does *not* appear in the class table. The auxiliary functions that look up fields and method declarations in the class table are equipped with special cases for `Object` that return the empty sequence of fields and the empty set of methods. (In full Java, the class `Object` does have several methods. We ignore these in FJ).

By looking at the class table, we can read off the subtype relation between classes. We write `C <: D` when `C` is a subtype of `D`, i.e., subtyping is the reflexive and transitive closure of the immediate subclass relation given by the `extends` clauses in CT. Formally, it is defined in the middle of Figure 1.

Expression typing:

$$\Gamma \vdash \mathtt{x} : \Gamma(\mathtt{x}) \tag{T-Var}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}_i : \mathtt{C}_i} \tag{T-Field}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \rightarrow \mathtt{C} \qquad \Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) : \mathtt{C}} \tag{T-Invk}$$

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}} \qquad \Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{new\ C}(\overline{\mathtt{e}}) : \mathtt{C}} \tag{T-New}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 : \mathtt{C}} \tag{T-UCast}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{C} <: \mathtt{D} \qquad \mathtt{C} \neq \mathtt{D}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 : \mathtt{C}} \tag{T-DCast}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \qquad \textit{stupid warning}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 : \mathtt{C}} \tag{T-SCast}$$

Method typing:

$$\frac{\begin{array}{c}\overline{\mathtt{x}} : \overline{\mathtt{C}}, \mathtt{this} : \mathtt{C} \vdash \mathtt{e}_0 : \mathtt{E}_0 \qquad \mathtt{E}_0 <: \mathtt{C}_0 \\ \mathtt{class\ C\ extends\ D\ \{...\}} \\ \text{if } \mathit{mtype}(\mathtt{m}, \mathtt{D}) = \overline{\mathtt{D}} \rightarrow \mathtt{D}_0, \text{ then } \overline{\mathtt{C}} = \overline{\mathtt{D}} \text{ and } \mathtt{C}_0 = \mathtt{D}_0\end{array}}{\mathtt{C}_0\ \mathtt{m}(\overline{\mathtt{C}}\ \overline{\mathtt{x}})\{\ \mathtt{return}\ \mathtt{e}_0;\ \}\ \mathtt{OK\ IN\ C}} \tag{T-Method}$$

Class typing:

$$\frac{\mathtt{K} = \mathtt{C}(\overline{\mathtt{D}}\ \overline{\mathtt{g}},\ \overline{\mathtt{C}}\ \overline{\mathtt{f}})\{\mathtt{super}(\overline{\mathtt{g}});\ \mathtt{this}.\overline{\mathtt{f}}\mathtt{=}\overline{\mathtt{f}};\} \qquad \mathit{fields}(\mathtt{D}) = \overline{\mathtt{D}}\ \overline{\mathtt{g}} \qquad \overline{\mathtt{M}}\ \mathtt{OK\ IN\ C}}{\mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\}\ \mathtt{OK}} \tag{T-Class}$$

Fig. 2. FJ: Typing rules.

The given class table is assumed to satisfy some sanity conditions: (1) $CT(\mathtt{C}) = \mathtt{class\ C}\ldots$ for every $\mathtt{C} \in dom(CT)$; (2) $\mathtt{Object} \notin dom(CT)$; (3) for every class name `C` (except `Object`) appearing anywhere in CT, we have $\mathtt{C} \in dom(CT)$; and (4) there are no cycles in the subtype relation induced by CT, i.e., the relation <: is antisymmetric. Given these conditions, we can identify a class table with a sequence of class declarations in an obvious way. Note that the types defined by the class table *are* allowed to be recursive, in the sense that the definition of a class `A` may use the name `A` in the types of its methods and instance variables. Indeed, even mutual recursion between class definitions is allowed.

For the typing and reduction rules, we need a few auxiliary definitions, given at the bottom of Figure 1. We write $\mathtt{m} \notin \bar{\mathtt{M}}$ to mean that the method definition of the name `m` is not included in $\bar{\mathtt{M}}$. The fields of a class `C`, written *fileds*(`C`), is a sequence $\bar{\mathtt{C}}\ \bar{\mathtt{f}}$ pairing the class of each field with its name, for all the fields declared in class `C` and all of its superclasses. The type of the method `m` in class `C`, written *mtype*(`m,C`), is a pair, written $\bar{\mathtt{B}} \to \mathtt{B}$, of a sequence of argument types $\bar{\mathtt{B}}$ and a result type `B`. (In Java proper, method body lookup is based not only on the method name but also on the static types of the actual arguments to deal with overloading, which we drop from FJ). Similarly, the body of the method `m` in class `C`, written *mbody*(`m,C`), is a pair, written $\bar{\mathtt{x}}.\mathtt{e}$, of a sequence of parameters $\bar{\mathtt{x}}$ and an expression `e`. Note that the functions *mtype*(`m,C`) and *mbody*(`m,C`) are both partial functions: since `Object` is assumed to have no methods in FJ, both *mtype*(`m,Object`) and *mbody*(`m,Object`) are undefined.

2.2 Typing

The typing rules for expressions, method declarations, and class declarations are in Figure 2. An environment Γ is a finite mapping from variables to types, written $\bar{\mathtt{x}}\mathtt{:}\bar{\mathtt{C}}$. The typing judgment for expressions has the form $\Gamma \vdash \mathtt{e} : \mathtt{C}$, read "in the environment Γ, expression `e` has type `C`." We abbreviate typing judgments on sequences in the obvious way, writing $\Gamma \vdash \bar{\mathtt{e}} : \bar{\mathtt{C}}$ as shorthand for $\Gamma \vdash \mathtt{e}_1 : \mathtt{C}_1, \ldots, \Gamma \vdash \mathtt{e}_n : \mathtt{C}_n$ and writing $\bar{\mathtt{C}} <: \bar{\mathtt{D}}$ as shorthand for $\mathtt{C}_1 <: \mathtt{D}_1, \ldots, \mathtt{C}_n <: \mathtt{D}_n$. The typing rules are syntax directed, with one rule for each form of expression, save that there are three rules for casts. Most of them are straightforward adaptations of the rules in Java; the typing rules for constructors and method invocations check that each actual parameter has a type that is a subtype of the corresponding formal parameter type.

One technical innovation in FJ is the introduction of "stupid" casts. There are three rules for type casts: in an *upcast* the subject is a subclass of the target; in a *downcast* the target is a subclass of the subject; and in a *stupid* cast the target is unrelated to the subject. The Java compiler rejects as ill typed an expression containing a stupid cast, but we must allow stupid casts in FJ if we are to formulate type soundness as a subject reduction theorem for a small-step semantics. This is because an expression without stupid casts may reduce to one containing a stupid cast. For example, consider the following, which uses classes `A` and `B` as defined in the previous section:

$$\mathtt{(A)}\underline{\mathtt{(Object)new\ B()}} \to \mathtt{(A)new\ B()}$$

We indicate the special nature of stupid casts by including the hypothesis *stupid warning* in the type rule for stupid casts (T-SCast); an FJ typing corresponds to a legal Java typing only if it does not contain this rule. (Stupid casts were omitted from Classic Java [Flatt et al. 1998a], causing its published proof of type soundness to be incorrect; this error was discovered independently by ourselves and the Classic Java authors).

The typing judgment for method declarations has the form `M OK IN C`, read "method declaration `M` is ok when it occurs in class `C`." It uses the expression typing judgment on the body of the method, where the free variables are the parameters of the method with their declared types, plus the special variable `this` with type `C`. (Thus, a method with a parameter of name `this` is not allowed, as the type environment is ill formed.) In case of overriding, if a method with the same name is declared in the superclass, then it must have the same type.

The typing judgment for class declarations has the form `L OK`, read "class declaration `L` is ok." It checks that the constructor applies `super` to the fields of the superclass and initializes the fields declared in this class, and that each method declaration in the class is ok.

The type of an expression may depend on the type of any methods it invokes, and the type of a method depends on the type of an expression (its body); so, it behooves us to check that there is no ill-defined circularity here. Indeed there is none: the circle is broken because the type of each method is explicitly declared. It is possible to load the class table and define the auxiliary functions *mtype*, *mbody*, and *fields* before all the classes in it are checked. Thus, each method body can independently typecheck, without inspecting the bodies of other methods it may invoke.

2.3 Reduction

The reduction relation is of the form $\mathtt{e} \rightarrow \mathtt{e}'$, read "expression `e` reduces to expression $\mathtt{e}'$ in one step." We write $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$.

The reduction rules are given in Figure 3. There are three reduction rules, one for field access, one for method invocation, and one for casting. These were already explained in the introduction to this section. We write $[\bar{\mathtt{d}}/\bar{\mathtt{x}}, \mathtt{e}/\mathtt{y}]\mathtt{e}_0$ for the result of replacing $\mathtt{x}_1$ by $\mathtt{d}_1, \ldots, \mathtt{x}_n$ by $\mathtt{d}_n$, and `y` by `e` in expression $\mathtt{e}_0$.

The reduction rules may be applied at any point in an expression, so we also need the obvious congruence rules (if $\mathtt{e} \rightarrow \mathtt{e}'$ then $\mathtt{e.f} \rightarrow \mathtt{e}'\mathtt{.f}$, and the like), which also appear in the figure.[2]

2.4 Properties

Formal definitions are fun, but the proof of the pudding is in ... well, the proof. If our definitions are sensible, we should be able to prove a type soundness result, which relates typing to computation. Indeed, we can prove such a result: if a term is well typed and it reduces to a normal form, then it is either a value of a subtype of the original term's type, or an expression that gets stuck at a downcast. The type-soundness theorem (Theorem 2.4.3) is proved by using the standard technique of subject reduction and progress theorems [Wright and Felleisen 1994].

THEOREM 2.4.1 (Subject Reduction). *If* $\Gamma \vdash \mathtt{e} : \mathtt{C}$ *and* $\mathtt{e} \rightarrow \mathtt{e}'$, *then* $\Gamma \vdash \mathtt{e}' : \mathtt{C}'$ *for some* $\mathtt{C}' <: \mathtt{C}$.

PROOF. See Appendix A.1. □

[2]We have chosen here to work with a nondeterministic reduction relation, similar to the full beta-reduction relation of the lambda-calculus. Naturally, more restricted reduction strategies can also be defined. For example, a call-by-value variant of FJ can be found in Pierce [2002].

Computation:

$$\frac{\textit{fields}(\mathtt{C}) = \bar{\mathtt{C}}\ \bar{\mathtt{f}}}{\mathtt{(new\ C(}\bar{\mathtt{e}}\mathtt{)).f}_i \longrightarrow \mathtt{e}_i} \qquad \text{(R-Field)}$$

$$\frac{\textit{mbody}(\mathtt{m}, \mathtt{C}) = \bar{\mathtt{x}}\mathtt{.e}_0}{\mathtt{(new\ C(}\bar{\mathtt{e}}\mathtt{)).m(}\bar{\mathtt{d}}\mathtt{)} \longrightarrow [\bar{\mathtt{d}}/\bar{\mathtt{x}}, \mathtt{new\ C(}\bar{\mathtt{e}}\mathtt{)/this}]\mathtt{e}_0} \qquad \text{(R-Invk)}$$

$$\frac{\mathtt{C} <: \mathtt{D}}{\mathtt{(D)(new\ C(}\bar{\mathtt{e}}\mathtt{))} \longrightarrow \mathtt{new\ C(}\bar{\mathtt{e}}\mathtt{)}} \qquad \text{(R-Cast)}$$

Congruence:

$$\frac{\mathtt{e}_0 \longrightarrow \mathtt{e}_0'}{\mathtt{e}_0\mathtt{.f} \longrightarrow \mathtt{e}_0'\mathtt{.f}} \qquad \text{(RC-Field)}$$

$$\frac{\mathtt{e}_0 \longrightarrow \mathtt{e}_0'}{\mathtt{e}_0\mathtt{.m(}\bar{\mathtt{e}}\mathtt{)} \longrightarrow \mathtt{e}_0'\mathtt{.m(}\bar{\mathtt{e}}\mathtt{)}} \qquad \text{(RC-Invk-Recv)}$$

$$\frac{\mathtt{e}_i \longrightarrow \mathtt{e}_i'}{\mathtt{e}_0\mathtt{.m(}\ldots\mathtt{,e}_i\mathtt{,}\ldots\mathtt{)} \longrightarrow \mathtt{e}_0\mathtt{.m(}\ldots\mathtt{,e}_i'\mathtt{,}\ldots\mathtt{)}} \qquad \text{(RC-Invk-Arg)}$$

$$\frac{\mathtt{e}_i \longrightarrow \mathtt{e}_i'}{\mathtt{new\ C(}\ldots\mathtt{,e}_i\mathtt{,}\ldots\mathtt{)} \longrightarrow \mathtt{new\ C(}\ldots\mathtt{,e}_i'\mathtt{,}\ldots\mathtt{)}} \qquad \text{(RC-New-Arg)}$$

$$\frac{\mathtt{e}_0 \longrightarrow \mathtt{e}_0'}{\mathtt{(C)e}_0 \longrightarrow \mathtt{(C)e}_0'} \qquad \text{(RC-Cast)}$$

Fig. 3. FJ: Reduction rules.

We can also show that if a program is well-typed, then the only way it can get stuck is if it reaches a point where it cannot perform a downcast.

THEOREM 2.4.2 (Progress). *Suppose* $\mathtt{e}$ *is a well-typed expression.*

(1) *If* $\mathtt{e}$ *includes* $\mathtt{new\ C_0(}\bar{\mathtt{e}}\mathtt{).f}$ *as a subexpression, then* $\textit{fields}(\mathtt{C}_0) = \bar{\mathtt{C}}\ \bar{\mathtt{f}}$ *and* $\mathtt{f} \in \bar{\mathtt{f}}$ *for some* $\bar{\mathtt{C}}$ *and* $\bar{\mathtt{f}}$.

(2) *If* $\mathtt{e}$ *includes* $\mathtt{new\ C_0(}\bar{\mathtt{e}}\mathtt{)\cdot m(}\bar{\mathtt{d}}\mathtt{)}$ *as a subexpression, then* $\textit{mbody}(\mathtt{m}, \mathtt{C}_0) = \bar{\mathtt{x}}\mathtt{.e}_0$ *and* $\#(\bar{\mathtt{x}}) = \#(\bar{\mathtt{d}})$ *for some* $\bar{\mathtt{x}}$ *and* $\mathtt{e}_0$.

PROOF. If $\mathtt{e}$ has $\mathtt{new\ C_0(}\bar{\mathtt{e}}\mathtt{).f}$ as a subexpression, then, by well-typedness of the subexpression, it is easy to check that *fields*($\mathtt{C}_0$) is well defined and $\mathtt{f}$ appears in it. Similarly, if $\mathtt{e}$ has $\mathtt{new\ C_0(}\bar{\mathtt{e}}\mathtt{).m(}\bar{\mathtt{d}}\mathtt{)}$ as a subexpression, then, it is also easy to show $\textit{mbody}(\mathtt{m}, \mathtt{C}) = \bar{\mathtt{x}}\mathtt{.e}_0$ and $\#(\bar{\mathtt{x}}) = \#(\bar{\mathtt{d}})$ from the fact that $\textit{mtype}(\mathtt{m}, \mathtt{C}) = \bar{\mathtt{C}} \rightarrow \mathtt{D}$ where $\#(\bar{\mathtt{x}}) = \#(\bar{\mathtt{C}})$. □

To state type soundness formally, we give the definition of values, given by the following syntax:

$$\mathtt{v} ::= \mathtt{new\ C(}\bar{\mathtt{v}}\mathtt{)}.$$

THEOREM 2.4.3 (FJ Type Soundness). *If* $\emptyset \vdash$ `e` : `C` *and* `e` $\rightarrow^*$ `e`$'$ *with* `e`$'$ *a normal form, then* `e`$'$ *is either a value* `v` *with* $\emptyset \vdash$ `v` : `D` *and* `D <: C`, *or an expression containing* `(D)new C(ē)` *where* `C <: D`.

PROOF. Immediate from Theorems 2.4.1 and 2.4.2. □

To state a similar property for casts, we say that an expression `e` is *cast-safe* in Γ if the type derivations of the underlying *CT* and $\Gamma \vdash$ `e` : `C` contain no downcasts or stupid casts (uses of rules T-DCast or T-SCast). In other words, a cast-safe program includes only upcasts. Then we see that a cast-safe expression always reduces to another cast-safe expression, and, moreover, typecasts in a cast-safe expression never fail, as shown in the following pair of theorems. (The proofs are straightforward).

THEOREM 2.4.4 (Reduction Preserves Cast-Safety). *If* `e` *is cast-safe in* Γ *and* `e` $\rightarrow$ `e`$'$, *then* `e`$'$ *is cast-safe in* Γ.

THEOREM 2.4.5 (Progress of Cast-Safe Programs). *Suppose* `e` *is cast-safe in* Γ. *If* `e` *has* `(C)new` $\texttt{C}_0$`(ē)` *as a subexpression, then* $\texttt{C}_0$ `<: C`.

COROLLARY 2.4.6 (No Typecast Errors in Cast-Safe Programs). *If* `e` *is cast-safe in* $\emptyset$ *and* `e` $\rightarrow^*$ `e`$'$ *with* `e`$'$ *a normal form, then* `e`$'$ *is a value* `v`.

3. FEATHERWEIGHT GJ

Just as GJ adds generic types to Java, Featherweight GJ (or FGJ, for short) adds generic types to FJ. Here is the class definition for pairs in FJ, rewritten with generic type parameters in FGJ.

```
class A extends Object {
  A() { super(); }
}
class B extends Object {
  B() { super(); }
}
class Pair<X extends Object, Y extends Object> extends Object {
  X fst;
  Y snd;
  Pair(X fst, Y snd) {
    super(); this.fst=fst; this.snd=snd;
  }
  <Z extends Object> Pair<Z,Y> setfst(Z newfst) {
    return new Pair<Z,Y>(newfst, this.snd);
  }
}
```

Both classes and methods may have generic type parameters. Here `X` and `Y` are parameters of the class, and `Z` is a parameter of the method `setfst`. Each type parameter has a *bound*; here `X`, `Y`, and `Z` are each bounded by `Object`.

In the context of the above definitions, the expression

```
new Pair<A,B>(new A(), new B()).setfst<B>(new B())
```

evaluates to the expression

```
new Pair<B,B>(new B(), new B())
```

If we were being extraordinarily pedantic, we would write `A<>` and `B<>` instead of `A` and `B`, but we allow the latter as an abbreviation for the former in order that FJ is a proper subset of FGJ.

In GJ, type parameters to generic method invocations are inferred. Thus, in GJ the expression above would be written

```
new Pair<A,B>(new A(), new B()).setfst(new B())
```

with no `<B>` in the invocation of `setfst`. So while FJ is a subset of Java, FGJ is not quite a subset of GJ. We regard FGJ as an intermediate language—the form that would result after type parameters have been inferred. (In fact, type arguments are not even optional in GJ: it is not allowed to supply explicit type arguments to a generic method, due to a parsing problem. For example, the GJ expression `e.m<A,B>(e′)` is parsed as the two expressions "`e.m < A`" and "`B > (e′)`", separated by a comma. One possible way to have control over inferred type arguments is to change the (static) types of (value) arguments by inserting upcasts on them; see the GJ paper by Bracha et al. [1998] for details.) While parameter inference is an important aspect of GJ, we chose in FGJ to concentrate on modeling other aspects of GJ.

The bound of a type variable may not be a type variable, but may be a type expression involving type variables, and may be recursive (or even, if there are several bounds, mutually recursive). For example, if `C<X>` and `D<Y>` are classes with one parameter each, one may have bounds such as `<X extends C<X>>` or even `<X extends C<Y>, Y extends D<X>>`. For more on bounds, including examples of the utility of recursive bounds, see the GJ paper by Bracha et al. [1998].

GJ and FGJ are intended to support either of two implementation styles. They may be implemented by *type-passing*, augmenting the runtime system to carry information about type parameters, or they may be implemented by *erasure*, removing all information about type parameters at runtime. This section explores the first style, giving a direct semantics for FGJ that maintains type parameters, and proving a type soundness theorem. Section 4 explores the second style, giving an erasure mapping from FGJ into FJ and showing a correspondence between reductions on FGJ expressions and reductions on FJ expressions. The second style corresponds to the current implementation of GJ, which compiles GJ into the Java Virtual Machine (JVM), which of course maintains no information about type parameters at runtime; the first style would correspond to using an augmented JVM that maintains information about type parameters.

Syntax:

$\mathtt{T} ::= \mathtt{X} \mid \mathtt{N}$

$\mathtt{N} ::= \mathtt{C{<}\bar{T}{>}}$

$\mathtt{L} ::= \mathtt{class\ C{<}\bar{X} \triangleleft \bar{N}{>} \triangleleft N\ \{\bar{T}\ \bar{f};\ K\ \bar{M}\}}$

$\mathtt{K} ::= \mathtt{C(\bar{T}\ \bar{f})\{super(\bar{f});\ this.\bar{f}{=}\bar{f};\}}$

$\mathtt{M} ::= \mathtt{{<}\bar{X} \triangleleft \bar{N}{>}\ T\ m(\bar{T}\ \bar{x})\{\ return\ e;\ \}}$

$\mathtt{e} ::= \mathtt{x} \mid \mathtt{e.f} \mid \mathtt{e.m{<}\bar{T}{>}(\bar{e})} \mid \mathtt{new\ N(\bar{e})} \mid \mathtt{(N)e}$

Fig. 4. FJ: Syntax.

3.1 Syntax

The abstract syntax of FGJ is given in Figure 4. In what follows, for the sake of conciseness we abbreviate the keyword `extends` to the symbol $\triangleleft$. The metavariables `X`, `Y`, and `Z` range over type variables; `S`, `T`, `U`, and `V` range over types; and `N`, `P`, and `Q` range over nonvariable types (types other than type variables). We write $\bar{\mathtt{X}}$ as shorthand for $\mathtt{X}_1,\ldots,\mathtt{X}_n$ (and similarly for $\bar{\mathtt{T}}$, $\bar{\mathtt{N}}$, etc.), and assume sequences of type variables contain no duplicate names. We allow `C<>` and `m<>` to be abbreviated as `C` and `m`, respectively.

As before, we assume a fixed class table *CT*, a mapping from class names `C` to class declarations `L` and the essentially same sanity conditions. (For condition (4), we use the relation $\mathtt{C} \trianglelefteq \mathtt{D}$ between class names, defined in Figure 5, as the reflexive and transitive closure induced by the clause $\mathtt{C{<}\bar{X} \triangleleft \bar{N}{>} \triangleleft D{<}\bar{T}{>}}$.)

As in FJ, for the typing and reduction rules, we need a few auxiliary definitions, given in Figure 5; these are fairly straightforward adaptations of the lookup rules given previously. The fields of a nonvariable type `N`, written *fields*(`N`), are a sequence of corresponding types and field names, $\bar{\mathtt{T}}\ \bar{\mathtt{f}}$. The type of the method invocation `m` at nonvariable type `N`, written *mtype*(`m`, `N`), is a type of the form $\mathtt{{<}\bar{X} \triangleleft \bar{N}{>}\bar{U}} \to \mathtt{U}$. In this form, the variables $\bar{\mathtt{X}}$ are bound in $\bar{\mathtt{N}}$, $\bar{\mathtt{U}}$, and `U`, and we regard α-convertible ones as equivalent; application of type substitution $[\bar{\mathtt{T}}/\bar{\mathtt{X}}]$ is defined in the customary manner. When $\bar{\mathtt{X}} \triangleleft \bar{\mathtt{N}}$ is empty, we abbreviate $\mathtt{{<}{>}\bar{U}} \to \mathtt{U}$ to $\bar{\mathtt{U}} \to \mathtt{U}$. The body of the method invocation `m` at nonvariable type `N` with type parameters $\bar{\mathtt{V}}$, written *mbody*($\mathtt{m{<}\bar{V}{>}}$, `N`), is a pair, written $\bar{\mathtt{x}}.\mathtt{e}$, of a sequence of parameters $\bar{\mathtt{x}}$ and an expression `e`.

3.2 Typing

An environment Γ is a finite mapping from variables to types, written $\bar{\mathtt{x}} : \bar{\mathtt{T}}$; a type environment Δ is a finite mapping from type variables to nonvariable types, written $\bar{\mathtt{X}} <: \bar{\mathtt{N}}$, which takes each type variable to its bound. The main judgments of the FGJ type system consist of one for subtyping $\Delta \vdash \mathtt{S} <: \mathtt{T}$, one for type well-formedness $\Delta \vdash \mathtt{T}$ ok, and one for typing $\Delta; \Gamma \vdash \mathtt{e} : \mathtt{T}$. We abbreviate a sequence of judgments in the obvious way: $\Delta \vdash \mathtt{S}_1 <: \mathtt{T}_1, \ldots, \Delta \vdash \mathtt{S}_n <: \mathtt{T}_n$ to $\Delta \vdash \bar{\mathtt{S}} <: \bar{\mathtt{T}}$; $\Delta \vdash \mathtt{T}_1$ ok, $\ldots$, $\Delta \vdash \mathtt{T}_n$ ok to $\Delta \vdash \bar{\mathtt{T}}$ ok; and $\Delta; \Gamma \vdash \mathtt{e}_1 : \mathtt{T}_1, \ldots, \Delta; \Gamma \vdash \mathtt{e}_n : \mathtt{T}_n$ to $\Delta; \Gamma \vdash \bar{\mathtt{e}} : \bar{\mathtt{T}}$.

Subclassing:

$$\texttt{C} \trianglelefteq \texttt{C} \qquad \frac{\texttt{C} \trianglelefteq \texttt{D} \quad \texttt{D} \trianglelefteq \texttt{E}}{\texttt{C} \trianglelefteq \texttt{E}} \qquad \frac{\texttt{class C<}\bar{\texttt{X}} \triangleleft \bar{\texttt{N}}\texttt{>} \triangleleft \texttt{D<}\bar{\texttt{T}}\texttt{> \{...\}}}{\texttt{C} \trianglelefteq \texttt{D}}$$

Field lookup:

$$\textit{fields}(\texttt{Object}) = \bullet \qquad \text{(F-Object)}$$

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

Method type lookup:

$$\frac{\begin{array}{c}\texttt{class C<}\bar{\texttt{X}} \triangleleft \bar{\texttt{N}}\texttt{>} \triangleleft \texttt{N \{}\bar{\texttt{S}}\ \bar{\texttt{f}}\texttt{; K }\bar{\texttt{M}}\texttt{\}} \\ \texttt{<}\bar{\texttt{Y}} \triangleleft \bar{\texttt{P}}\texttt{> U m(}\bar{\texttt{U}}\ \bar{\texttt{x}}\texttt{)\{ return e; \}} \in \bar{\texttt{M}}\end{array}}{\textit{mtype}(\texttt{m}, \texttt{C<}\bar{\texttt{T}}\texttt{>}) = [\bar{\texttt{T}}/\bar{\texttt{X}}](\texttt{<}\bar{\texttt{Y}} \triangleleft \bar{\texttt{P}}\texttt{>}\bar{\texttt{U}} \rightarrow \texttt{U})} \qquad \text{(MT-Class)}$$

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

Method body lookup:

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

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

Fig. 5. FGJ: Auxiliary functions.

Bounds of types. We write $\textit{bound}_\Delta(\texttt{T})$ for the upper bound of T in Δ, as defined in Figure 6. Unlike calculi such as $F_\leq$ [Cardelli et al. 1994], this promotion relation does not need to be defined recursively: the bound of a type variable is always a nonvariable type.

Subtyping. The subtyping relation $\Delta \vdash \texttt{S} <: \texttt{T}$, read as "S is subtype of T in Δ," is defined in Figure 6. As before, subtyping is the reflexive and transitive closure of the extends relation. Type parameters are *invariant* with regard to subtyping (for the usual reasons; a type parameter can be both argument and result type of one method), so $\Delta \vdash \bar{\texttt{T}} <: \bar{\texttt{U}}$ does *not* imply $\Delta \vdash \texttt{C<}\bar{\texttt{T}}\texttt{>} <: \texttt{C<}\bar{\texttt{U}}\texttt{>}$.

Well-formed types. If the declaration of a class C begins class $\texttt{C<}\bar{\texttt{X}} \triangleleft \bar{\texttt{N}}\texttt{>}$, then a type like $\texttt{C<}\bar{\texttt{T}}\texttt{>}$ is well formed only if substituting $\bar{\texttt{T}}$ for $\bar{\texttt{X}}$ respects the bounds $\bar{\texttt{N}}$, i.e., if $\bar{\texttt{T}} <: [\bar{\texttt{T}}/\bar{\texttt{X}}]\bar{\texttt{N}}$. We write $\Delta \vdash \texttt{T}$ ok if type T is well formed in context Δ. The rules for well-formed types appear in the middle of Figure 6. Note that we perform a simultaneous substitution, so any variable in $\bar{\texttt{X}}$ may appear in $\bar{\texttt{N}}$, permitting recursion and mutual recursion between variables and bounds.

Bound of type:

$$\begin{aligned} \mathit{bound}_\Delta(\texttt{X}) &= \Delta(\texttt{X}) \\ \mathit{bound}_\Delta(\texttt{N}) &= \texttt{N} \end{aligned}$$

Subtyping:

$$\Delta \vdash \texttt{T} <: \texttt{T} \qquad \text{(S-Refl)}$$

$$\frac{\Delta \vdash \texttt{S} <: \texttt{T} \qquad \Delta \vdash \texttt{T} <: \texttt{U}}{\Delta \vdash \texttt{S} <: \texttt{U}} \qquad \text{(S-Trans)}$$

$$\Delta \vdash \texttt{X} <: \Delta(\texttt{X}) \qquad \text{(S-Var)}$$

$$\frac{\texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{N \{...\}}}{\Delta \vdash \texttt{C<}\overline{\texttt{T}}\texttt{>} <: [\overline{\texttt{T}}/\overline{\texttt{X}}]\texttt{N}} \qquad \text{(S-Class)}$$

Well-formed types:

$$\Delta \vdash \texttt{Object}\ \text{ok} \qquad \text{(WF-Object)}$$

$$\frac{\texttt{X} \in \mathit{dom}(\Delta)}{\Delta \vdash \texttt{X}\ \text{ok}} \qquad \text{(WF-Var)}$$

$$\frac{\begin{array}{c}\texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{N \{...\}} \\ \Delta \vdash \overline{\texttt{T}}\ \text{ok} \qquad \Delta \vdash \overline{\texttt{T}} <: [\overline{\texttt{T}}/\overline{\texttt{X}}]\overline{\texttt{N}}\end{array}}{\Delta \vdash \texttt{C<}\overline{\texttt{T}}\texttt{>}\ \text{ok}} \qquad \text{(WF-Class)}$$

Valid downcast:

$$\frac{\mathit{dcast}(\texttt{C}, \texttt{D}) \qquad \mathit{dcast}(\texttt{D}, \texttt{E})}{\mathit{dcast}(\texttt{C}, \texttt{E})} \qquad\qquad \frac{\begin{array}{c}\texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{D<}\overline{\texttt{T}}\texttt{> \{...\}} \\ \overline{\texttt{X}} = \mathit{FV}(\overline{\texttt{T}})\end{array}}{\mathit{dcast}(\texttt{C}, \texttt{D})}$$

($\mathit{FV}(\overline{\texttt{T}})$ denotes the set of type variables in $\overline{\texttt{T}}$.)

Valid method overriding:

$$\frac{\mathit{mtype}(\texttt{m}, \texttt{N}) = \texttt{<}\overline{\texttt{Z}} \triangleleft \overline{\texttt{Q}}\texttt{>}\overline{\texttt{U}} \rightarrow \texttt{U}_0 \text{ implies } \overline{\texttt{P}}, \overline{\texttt{T}} = [\overline{\texttt{Y}}/\overline{\texttt{Z}}](\overline{\texttt{Q}}, \overline{\texttt{U}}) \text{ and } \overline{\texttt{Y}} <: \overline{\texttt{P}} \vdash \texttt{T}_0 <: [\overline{\texttt{Y}}/\overline{\texttt{Z}}]\texttt{U}_0}{\mathit{override}(\texttt{m}, \texttt{N}, \texttt{<}\overline{\texttt{Y}} \triangleleft \overline{\texttt{P}}\texttt{>}\overline{\texttt{T}} \rightarrow \texttt{T}_0)}$$

Fig. 6. FGJ: Subtyping and type well-formedness rules.

A type environment Δ is well formed if $\Delta \vdash \Delta(\texttt{X})$ ok for all $\texttt{X}$ in $\mathit{dom}(\Delta)$. We also say that an environment Γ is well formed with respect to Δ, written $\Delta \vdash \Gamma$ ok, if $\Delta \vdash \Gamma(\texttt{x})$ ok for all $\texttt{x}$ in $\mathit{dom}(\Gamma)$.

Typing rules. Typing rules for expressions, methods, and classes appear in Figure 7. The typing judgment for expressions is of the form $\Delta; \Gamma \vdash \texttt{e}:\texttt{T}$, read as "in the type environment Δ and the environment Γ, the expression $\texttt{e}$ has

Expression typing:

$$\Delta;\Gamma \vdash \texttt{x} : \Gamma(\texttt{x}) \qquad \text{(GT-Var)}$$

$$\frac{\Delta;\Gamma \vdash \texttt{e}_0 : \texttt{T}_0 \qquad \mathit{fields}(\mathit{bound}_\Delta(\texttt{T}_0)) = \overline{\texttt{T}}\ \overline{\texttt{f}}}{\Delta;\Gamma \vdash \texttt{e}_0\texttt{.f}_i : \texttt{T}_i} \qquad \text{(GT-Field)}$$

$$\frac{\begin{array}{c}\Delta;\Gamma \vdash \texttt{e}_0 : \texttt{T}_0 \qquad \mathit{mtype}(\texttt{m},\ \mathit{bound}_\Delta(\texttt{T}_0)) = \texttt{<}\overline{\texttt{Y}} \triangleleft \overline{\texttt{P}}\texttt{>}\overline{\texttt{U}}\to\texttt{U} \\ \Delta \vdash \overline{\texttt{V}} \text{ ok} \qquad \Delta \vdash \overline{\texttt{V}} <: [\overline{\texttt{V}}/\overline{\texttt{Y}}]\overline{\texttt{P}} \qquad \Delta;\Gamma \vdash \overline{\texttt{e}} : \overline{\texttt{S}} \qquad \Delta \vdash \overline{\texttt{S}} <: [\overline{\texttt{V}}/\overline{\texttt{Y}}]\overline{\texttt{U}}\end{array}}{\Delta;\Gamma \vdash \texttt{e}_0\texttt{.m<}\overline{\texttt{V}}\texttt{>(}\overline{\texttt{e}}\texttt{)} : [\overline{\texttt{V}}/\overline{\texttt{Y}}]\texttt{U}} \qquad \text{(GT-Invk)}$$

$$\frac{\Delta \vdash \texttt{N} \text{ ok} \qquad \mathit{fields}(\texttt{N}) = \overline{\texttt{T}}\ \overline{\texttt{f}} \qquad \Delta;\Gamma \vdash \overline{\texttt{e}} : \overline{\texttt{S}} \qquad \Delta \vdash \overline{\texttt{S}} <: \overline{\texttt{T}}}{\Delta;\Gamma \vdash \texttt{new N(}\overline{\texttt{e}}\texttt{)} : \texttt{N}} \qquad \text{(GT-New)}$$

$$\frac{\Delta;\Gamma \vdash \texttt{e}_0 : \texttt{T}_0 \qquad \Delta \vdash \mathit{bound}_\Delta(\texttt{T}_0) <: \texttt{N}}{\Delta;\Gamma \vdash \texttt{(N)e}_0 : \texttt{N}} \qquad \text{(GT-UCast)}$$

$$\frac{\begin{array}{c}\Delta;\Gamma \vdash \texttt{e}_0 : \texttt{T}_0 \qquad \Delta \vdash \texttt{N} \text{ ok} \qquad \Delta \vdash \texttt{N} <: \mathit{bound}_\Delta(\texttt{T}_0) \\ \texttt{N} = \texttt{C<}\overline{\texttt{T}}\texttt{>} \qquad \mathit{bound}_\Delta(\texttt{T}_0) = \texttt{D<}\overline{\texttt{U}}\texttt{>} \qquad \mathit{dcast}(\texttt{C},\ \texttt{D})\end{array}}{\Delta;\Gamma \vdash \texttt{(N)e}_0 : \texttt{N}} \qquad \text{(GT-DCast)}$$

$$\frac{\begin{array}{c}\Delta;\Gamma \vdash \texttt{e}_0 : \texttt{T}_0 \qquad \Delta \vdash \texttt{N} \text{ ok} \qquad \texttt{N} = \texttt{C<}\overline{\texttt{T}}\texttt{>} \qquad \mathit{bound}_\Delta(\texttt{T}_0) = \texttt{D<}\overline{\texttt{U}}\texttt{>} \\ \texttt{C} \not\trianglelefteq \texttt{D} \qquad \texttt{D} \not\trianglelefteq \texttt{C} \qquad \mathit{stupid\ warning}\end{array}}{\Delta;\Gamma \vdash \texttt{(N)e}_0 : \texttt{N}} \qquad \text{(GT-SCast)}$$

Method typing:

$$\frac{\begin{array}{c}\Delta = \overline{\texttt{X}} <: \overline{\texttt{N}},\ \overline{\texttt{Y}} <: \overline{\texttt{P}} \qquad \Delta \vdash \overline{\texttt{T}},\ \texttt{T},\ \overline{\texttt{P}} \text{ ok} \\ \Delta;\overline{\texttt{x}} : \overline{\texttt{T}},\ \texttt{this} : \texttt{C<}\overline{\texttt{X}}\texttt{>} \vdash \texttt{e}_0 : \texttt{S} \qquad \Delta \vdash \texttt{S} <: \texttt{T} \\ \texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{N \{...\}} \qquad \mathit{override}(\texttt{m},\ \texttt{N},\ \texttt{<}\overline{\texttt{Y}} \triangleleft \overline{\texttt{P}}\texttt{>}\overline{\texttt{T}}\to\texttt{T})\end{array}}{\texttt{<}\overline{\texttt{Y}} \triangleleft \overline{\texttt{P}}\texttt{> T m(}\overline{\texttt{T}}\ \overline{\texttt{x}}\texttt{)\{ return e}_0\texttt{; \} OK IN C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>}} \qquad \text{(GT-Method)}$$

Class typing:

$$\frac{\begin{array}{c}\overline{\texttt{X}} <: \overline{\texttt{N}} \vdash \overline{\texttt{N}},\ \texttt{N},\ \overline{\texttt{T}} \text{ ok} \qquad \mathit{fields}(\texttt{N}) = \overline{\texttt{U}}\ \overline{\texttt{g}} \qquad \overline{\texttt{M}} \texttt{ OK IN C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \\ \texttt{K} = \texttt{C(}\overline{\texttt{U}}\ \overline{\texttt{g}}\texttt{, }\overline{\texttt{T}}\ \overline{\texttt{f}}\texttt{)\{super(}\overline{\texttt{g}}\texttt{); this.}\overline{\texttt{f}}\texttt{ = }\overline{\texttt{f}}\texttt{;\}}\end{array}}{\texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{N \{}\overline{\texttt{T}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\} OK}} \qquad \text{(GT-Class)}$$

Fig. 7. FGJ: Typing rules.

type T.” Most of the subtleties are in the field and method lookup relations that we have already seen; the typing rules themselves are straightforward.

In the rule GT-DCast, the last premise *dcast*(C, D) ensures that the result of the cast will be the same at runtime, no matter whether we use the high-level (type-passing) reduction rules defined later in this section or the erasure semantics considered in Section 4. Intuitively, when $\texttt{C<}\overline{\texttt{T}}\texttt{>} <: \texttt{D<}\overline{\texttt{U}}\texttt{>}$ holds, all the type arguments $\overline{\texttt{T}}$ of C must “contribute” for the relation to hold. For example, suppose we have defined the following two classes:

```
class List<X ◁ Object> ◁ Object {...}
class LinkedList<X ◁ Object> ◁ List<X> {...}
```

Now, if `o` has type `Object`, then the cast `(List<C>)o` is not permitted. (If, at run-time, `o` is bound to `new List<D>()`, then the cast would fail in the type-passing semantics but succeed in the erasure semantics, since `(List<C>)o` erases to `(List)o` while both `new List<C>()` and `new List<D>()` erase to `new List()`.) On the other hand, if `cl` has type `List<C>`, then the cast `(LinkedList<C>)cl` is permitted, since the type-passing and erased versions of the cast are guaranteed to either both succeed or both fail. The formal definition of *dcast*(C, D) appears in Figure 6. (In GJ, *raw types* are provided to overcome the lack of expressiveness caused by this restriction. In the above example, programmers could write an expression like `(List)o`, instead of `(List<C>)o`, though type argument information is lost at that point; here, the type `List` is called the raw type from the class `List`. For simplicity, we do not model raw types in this article and are currently working on them [Igarashi et al. 2001].)

The typing rule for methods contains one additional subtlety. In FGJ (and GJ), unlike in FJ (and Java), covariant overriding on the method result type is allowed (see the rule for valid method overriding at the bottom of Figure 6), i.e., the result type of a method may be a subtype of the result type of the corresponding method in the superclass, although the bounds of type variables and the argument types must be identical (modulo renaming of type variables).

As before, a class table is ok if all its class definitions are ok.

3.3 Reduction

The operational semantics of FGJ programs is only a little more complicated than what we had in FJ. The rules appear in Figure 8. In the rule GR-CAST, the empty environment ∅ indicates the fact that whether or not `N` is a subtype of `P` must be checked without information on runtime type arguments.

3.4 Properties

Type Soundness. FGJ programs enjoy subject reduction, progress properties, and thus a type soundness property exactly like programs in FJ (Theorems 3.4.1, 3.4.2, and 3.4.3), The basic structures of the proofs are similar to those of Theorems 2.4.1 and 2.4.2. For subject reduction, however, since we now have parametric polymorphism combined with subtyping, we need a few more lemmas The main lemmas required are a term substitution lemma as before, plus similar lemmas about the preservation of subtyping and typing under *type* substitution. (Readers familiar with proofs of subject reduction for typed lambda-calculi like $F_{\leq}$ [Cardelli et al. 1994] will notice many similarities). The required lemmas include three substitution lemmas, which are proved by straightforward induction on a derivation of $\Delta \vdash \mathtt{S} <: \mathtt{T}$ or $\Delta; \Gamma \vdash \mathtt{e} : \mathtt{T}$. In the following proof, the underlying class table is assumed to be ok.

THEOREM 3.4.1 (Subject Reduction). *If* $\Delta; \Gamma \vdash \mathtt{e} : \mathtt{T}$ *and* $\mathtt{e} \rightarrow \mathtt{e}'$, *then* $\Delta; \Gamma \vdash \mathtt{e}' : \mathtt{T}'$, *for some* $\mathtt{T}'$ *such that* $\Delta \vdash \mathtt{T}' <: \mathtt{T}$.

Computation:

$$\frac{\textit{fields}(\texttt{N}) = \bar{\texttt{T}}\ \bar{\texttt{f}}}{(\texttt{new N(}\bar{\texttt{e}}\texttt{)).f}_i \longrightarrow \texttt{e}_i} \quad \text{(GR-Field)}$$

$$\frac{\textit{mbody}(\texttt{m<}\bar{\texttt{V}}\texttt{>}, \texttt{N}) = \bar{\texttt{x}}.\texttt{e}_0}{(\texttt{new N(}\bar{\texttt{e}}\texttt{)).m<}\bar{\texttt{V}}\texttt{>(}\bar{\texttt{d}}\texttt{)} \longrightarrow [\bar{\texttt{d}}/\bar{\texttt{x}}, \texttt{new N(}\bar{\texttt{e}}\texttt{)/this}]\texttt{e}_0} \quad \text{(GR-Invk)}$$

$$\frac{\emptyset \vdash \texttt{N} <: \texttt{P}}{\texttt{(P)(new N(}\bar{\texttt{e}}\texttt{))} \longrightarrow \texttt{new N(}\bar{\texttt{e}}\texttt{)}} \quad \text{(GR-Cast)}$$

Congruence:

$$\frac{\texttt{e}_0 \longrightarrow \texttt{e}_0{}'}{\texttt{e}_0\texttt{.f} \longrightarrow \texttt{e}_0{}'\texttt{.f}} \quad \text{(GRC-Field)}$$

$$\frac{\texttt{e}_0 \longrightarrow \texttt{e}_0{}'}{\texttt{e}_0\texttt{.m<}\bar{\texttt{T}}\texttt{>(}\bar{\texttt{e}}\texttt{)} \longrightarrow \texttt{e}_0{}'\texttt{.m<}\bar{\texttt{T}}\texttt{>(}\bar{\texttt{e}}\texttt{)}} \quad \text{(GRC-Inv-Recv)}$$

$$\frac{\texttt{e}_i \longrightarrow \texttt{e}_i{}'}{\texttt{e}_0\texttt{.m<}\bar{\texttt{T}}\texttt{>(}\ldots\texttt{,e}_i\texttt{,}\ldots\texttt{)} \longrightarrow \texttt{e}_0\texttt{.m<}\bar{\texttt{T}}\texttt{>(}\ldots\texttt{e}_i{}'\texttt{,}\ldots\texttt{)}} \quad \text{(GRC-Inv-Arg)}$$

$$\frac{\texttt{e}_i \longrightarrow \texttt{e}_i{}'}{\texttt{new N(}\ldots\texttt{,e}_i\texttt{,}\ldots\texttt{)} \longrightarrow \texttt{new N(}\ldots\texttt{e}_i{}'\texttt{,}\ldots\texttt{)}} \quad \text{(GRC-New-Arg)}$$

$$\frac{\texttt{e}_0 \longrightarrow \texttt{e}_0{}'}{\texttt{(N)e}_0 \longrightarrow \texttt{(N)e}_0{}'} \quad \text{(GRC-Cast)}$$

Fig. 8. FGJ: Reduction rules.

PROOF. See Appendix A.2. □

THEOREM 3.4.2 (Progress). *Suppose* e *is a well-typed expression.*

(1) *If* e *includes* $\texttt{new N}_0\texttt{(}\bar{\texttt{e}}\texttt{).f}$ *as a subexpression, then* $\textit{fields}(\texttt{N}_0) = \bar{\texttt{T}}\ \bar{\texttt{f}}$ *and* $\texttt{f} \in \bar{\texttt{f}}$ *for some* $\bar{\texttt{T}}$ *and* $\bar{\texttt{f}}$.

(2) *If* e *includes* $\texttt{new N}_0\texttt{(}\bar{\texttt{e}}\texttt{).m<}\bar{\texttt{V}}\texttt{>(}\bar{\texttt{d}}\texttt{)}$ *as a subexpression, then* $\textit{mbody}(\texttt{m<}\bar{\texttt{V}}\texttt{>}, \texttt{N}_0) = \bar{\texttt{x}}.\texttt{e}_0$ *and* $\#(\bar{\texttt{x}}) = \#(\bar{\texttt{d}})$ *for some* $\bar{\texttt{x}}$ *and* $\texttt{e}_0$.

PROOF. Similar to the proof of Theorem 2.4.2. □

As we did for FJ, we will give the definition of FGJ values below, to state FGJ type soundness formally:

$$\texttt{w} ::= \texttt{new N(}\bar{\texttt{w}}\texttt{)}.$$

THEOREM 3.4.3 (FGJ Type Soundness). *If* $\emptyset;\emptyset \vdash \texttt{e} : \texttt{T}$ *and* $\texttt{e} \rightarrow^* \texttt{e}'$ *with* $\texttt{e}'$ *a normal form, then* $\texttt{e}'$ *is either* (1) *an FGJ value* w *with* $\emptyset;\emptyset \vdash \texttt{w} : \texttt{S}$ *and* $\emptyset \vdash \texttt{S} <: \texttt{T}$ *or* (2) *an expression containing* $\texttt{(P)new N(}\bar{\texttt{e}}\texttt{)}$ *where* $\emptyset \vdash \texttt{N} <: \texttt{P}$.

PROOF. Immediate from Theorems 3.4.1 and 3.4.2. □

Backward compatibility. FGJ is backward compatible with FJ. Intuitively, this means that an implementation of FGJ can be used to typecheck and execute FJ programs without changing their meaning. In the following statements, we use subscripts FJ or FGJ to show which set of rules is used.

LEMMA 3.4.4. *If CT is an FJ class table, then* $fields_{\mathrm{FJ}}(\mathtt{C}) = fields_{\mathrm{FGJ}}(\mathtt{C})$ *for all* $\mathtt{C} \in dom(CT)$.

LEMMA 3.4.5. *Suppose CT is an FJ class table. Then,* $mtype_{\mathrm{FJ}}(\mathtt{m}, \mathtt{C}) = \bar{\mathtt{C}} \to \mathtt{C}$ *if and only if* $mtype_{\mathrm{FGJ}}(\mathtt{m}, \mathtt{C}) = \bar{\mathtt{C}} \to \mathtt{C}$. *Similarly,* $mbody_{\mathrm{FJ}}(\mathtt{m}, \mathtt{C}) = \bar{\mathtt{x}}.\mathtt{e}$ *if and only if* $mbody_{\mathrm{FGJ}}(\mathtt{m}, \mathtt{C}) = \bar{\mathtt{x}}.\mathtt{e}$.

PROOF. Both lemmas are easy. Note that in an FJ class table all substitutions in the derivations are empty and that there are no polymorphic methods. □

We can show that a well-typed FJ program is always a well-typed FGJ program and that FJ and FGJ reduction correspond. (Note that it is not quite the case that the well-typedness of an FJ program under the FGJ rules implies its well-typedness in FJ, because FGJ allows covariant overriding and FJ does not. In other words, FGJ is not a conservative extension of FJ).

THEOREM 3.4.6 (Backward Compatibility). *If an FJ program* $(\mathtt{e}, CT)$ *is well typed under the typing rules of FJ, then it is also well typed under the rules of FGJ. Moreover, for all FJ programs* $\mathtt{e}$ *and* $\mathtt{e}'$ *(whether well typed or not),* $\mathtt{e} \to_{\mathrm{FJ}} \mathtt{e}'$ *if and only if* $\mathtt{e} \to_{\mathrm{FGJ}} \mathtt{e}'$.

PROOF. The first half is shown by straightforward induction on the derivation of $\Gamma \vdash \mathtt{e} : \mathtt{C}$ (using FJ typing rules), followed by an analysis of the rules T-METHOD and T-CLASS. In the proof of the second half, both directions are shown by induction on a derivation of the reduction relation, with a case analysis on the last rule used. □

4. COMPILING FGJ TO FJ

We now explore the second implementation style for GJ and FGJ. The current GJ compiler works by translation into the standard JVM, which maintains no information about type parameters at runtime. We model this compilation in our framework by an *erasure* translation from FGJ into FJ. We show that this translation maps well-typed FGJ programs into well-typed FJ programs, and that the behavior of a program in FGJ matches (in a suitable sense) the behavior of its erasure under the FJ reduction rules.

A program is erased by replacing types with their erasures, inserting downcasts where required. A type is erased by removing type parameters, and replacing type variables with the erasure of their bounds. For example, the class `Pair<X,Y>` in the previous section erases to the following:

```
class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst=fst; this.snd=snd;
   }
   Pair setfst(Object newfst) {
     return new Pair(newfst, this.snd);
   }
 }
```

Similarly, the field selection

```
new Pair<A,B>(new A(), new B()).snd
```

erases to

```
(B)new Pair(new A(), new B()).snd
```

where the added downcast `(B)` recovers type information of the original program. We call such downcasts inserted by erasure *synthetic.* A key property of the erasure transformation is that it satisfies a so-called *cast-iron guarantee*: if the FGJ program is well typed, then no downcast inserted by the erasure transformation will fail at runtime. In the following discussion, we often distinguish synthetic casts from typecasts derived from original FGJ programs by superscripting typecast expressions, writing $(\mathtt{C})^{s}\mathtt{e}$. Otherwise, they behave exactly the same as ordinary typecasts.

4.1 Erasure of Types

To erase a type, we remove any type parameters and replace type variables with the erasure of their bounds. Write $|\mathtt{T}|_{\Delta}$ for the erasure of type `T` with respect to type environment Δ, defined by

$$|\mathtt{T}|_{\Delta} = \mathtt{C}$$

where $\mathit{bound}_{\Delta}(\mathtt{T}) = \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$.

4.2 Field and Method Lookup

In FGJ (and GJ), a subclass may extend an instantiated superclass. This means that, unlike in FJ (and Java), the types of the fields and the methods in the subclass may not be identical to the types in the superclass. In order to specify a type-preserving erasure from FGJ to FJ, it is necessary to define additional auxiliary functions that look up the type of a field or method in the *highest* superclass in which it is defined.

For example, consider a slight variant of the generic class `Pair<X,Y>`, where the method `setfst` is not declared to be polymorphic, taking an argument of the same element type `X`:

```
class Pair<X extends Object, Y extends Object> extends Object {
  X fst;  Y snd;
  Pair(X fst, Y snd) {
    super(); this.fst=fst; this.snd=snd;
  }
  Pair<X,Y> setfst(X newfst) {
      return new Pair<X,Y>(newfst, this.snd);
    }
  }
```

Note that the erasure of this class is the same as above. Then, a subclass `PairOfA`, declared below as a subclass of the instantiation `Pair<A,A>`, instantiates both `X` and `Y`.

```
  class PairOfA extends Pair<A,A> {
    PairOfA(A fst, A snd) { super(fst, snd); }
    PairOfA setfst(A newfst) {
      return new PairOfA(newfst, this.snd);
    }
  }
```

In the `setfst` method, the argument type `A` matches the argument type of `setfst` in `Pair<A,A>`, while the result type `PairOfA` is a subtype of the result type in `Pair<A,A>`; this is permitted by FGJ's covariant subtyping, as discussed in the previous section. Erasing the class `PairOfA` yields the following:

```
  class PairOfA extends Pair {
    PairOfA(Object fst, Object snd) { super(fst, snd); }
    Pair setfst(Object newfst) {
      return new PairOfA((A)newfst, (A)this.snd);
    }
  }
```

Here, arguments to the constructor and the method are given type `Object`, even though the erasure of `A` is itself; and the result of the method is given type `Pair`, even though the erasure of `PairOfA` is itself. In both cases, the types are chosen to correspond to types in `Pair`, the highest superclass in which the fields and methods are defined. Notice that the synthetic cast `(A)` is inserted at where the parameter `newfst` appears: it is required to recover type information of the original program, as well as the one at `this.snd`.

We define variants of the auxiliary functions that find the types of fields and methods in the highest superclass in which they are defined. The maximum field types of a class `C`, written *fieldsmax*(`C`), is the sequence of pairs of a type and a field name defined as follows:

$$\mathit{fieldsmax}(\texttt{Object}) = \bullet$$

$$\frac{\begin{array}{c}\texttt{class C<}\bar{\texttt{X}} \lhd \bar{\texttt{N}}\texttt{>} \lhd \texttt{D<}\bar{\texttt{U}}\texttt{> \{}\bar{\texttt{T}}\ \bar{\texttt{f}}\texttt{; ... \}}\\ \Delta = \bar{\texttt{X}}\texttt{<:}\bar{\texttt{N}} \qquad \bar{\texttt{C}}\ \bar{\texttt{g}} = \mathit{fieldsmax}(\texttt{D})\end{array}}{\mathit{fieldsmax}(\texttt{C}) = \bar{\texttt{C}}\ \bar{\texttt{g}}, |\bar{\texttt{T}}|_\Delta\ \bar{\texttt{f}}}$$

The maximum method type of `m` in `C`, written *mtypemax*(`m`, `C`), is defined as follows:

$$\frac{\texttt{class C<}\bar{\texttt{X}} \lhd \bar{\texttt{N}}\texttt{>} \lhd \texttt{D<}\bar{\texttt{U}}\texttt{> \{...\}} \qquad \texttt{<}\bar{\texttt{Y}} \lhd \bar{\texttt{P}}\texttt{>}\bar{\texttt{T}} \to \texttt{T} = \mathit{mtype}(\texttt{m}, \texttt{D<}\bar{\texttt{U}}\texttt{>})}{\mathit{mtypemax}(\texttt{m}, \texttt{C}) = \mathit{mtypemax}(\texttt{m}, \texttt{D})}$$

$$\frac{\begin{array}{c}\texttt{class C<}\bar{\texttt{X}}\triangleleft\bar{\texttt{N}}\texttt{>}\triangleleft\texttt{D<}\bar{\texttt{U}}\texttt{> \{... }\bar{\texttt{M}}\texttt{ \}} \qquad \mathit{mtype}(\texttt{m},\texttt{D<}\bar{\texttt{U}}\texttt{>})\text{ undefined}\\ \texttt{<}\bar{\texttt{Y}}\triangleleft\bar{\texttt{P}}\texttt{> T m(}\bar{\texttt{T}}\ \bar{\texttt{x}}\texttt{)\{ return e; \}}\in\bar{\texttt{M}} \qquad \Delta=\bar{\texttt{X}}\texttt{<:}\,\bar{\texttt{N}},\bar{\texttt{Y}}\texttt{<:}\,\bar{\texttt{P}}\end{array}}{\mathit{mtypemax}(\texttt{m},\texttt{C}) = |\bar{\texttt{T}}|_\Delta \to |\texttt{T}|_\Delta}$$

We also need a way to look up the maximum type of a given field. If *fieldsmax*(C) $= \bar{\texttt{D}}\ \bar{\texttt{f}}$, then we set *fieldsmax*(C) $(\texttt{f}_i) = \texttt{D}_i$.

4.3 Erasure of Expressions

The erasure of an expression depends on the typing of that expression, since the types are used to determine which downcasts to insert. The erasure rules are optimized to omit casts when it is trivially safe to do so; this happens when the maximum type is equal to the erased type.

Write $|\texttt{e}|_{\Delta,\Gamma}$ for the erasure of a well-typed expression e with respect to environment Γ and type environment Δ:

$$|\texttt{x}|_{\Delta,\Gamma} = \texttt{x} \qquad \text{(E-VAR)}$$

$$\frac{\Delta;\Gamma\vdash \texttt{e}_0\texttt{.f}:\texttt{T} \qquad \Delta;\Gamma\vdash\texttt{e}_0:\texttt{T}_0 \qquad \mathit{fieldsmax}(|\texttt{T}_0|_\Delta)(\texttt{f}) = |\texttt{T}|_\Delta}{|\texttt{e}_0\texttt{.f}|_{\Delta,\Gamma} = |\texttt{e}_0|_{\Delta,\Gamma}\texttt{.f}} \qquad \text{(E-FIELD)}$$

$$\frac{\Delta;\Gamma\vdash \texttt{e}_0\texttt{.f}:\texttt{T} \qquad \Delta;\Gamma\vdash\texttt{e}_0:\texttt{T}_0 \qquad \mathit{fieldsmax}(|\texttt{T}_0|_\Delta)(\texttt{f}) \neq |\texttt{T}|_\Delta}{|\texttt{e}_0\texttt{.f}|_{\Delta,\Gamma} = (|\texttt{T}|_\Delta)^s|\texttt{e}_0|_{\Delta,\Gamma}\texttt{.f}} \qquad \text{(E-FIELD-CAST)}$$

$$\frac{\Delta;\Gamma\vdash \texttt{e}_0\texttt{.m<}\bar{\texttt{V}}\texttt{>(}\bar{\texttt{e}}\texttt{)}:\texttt{T} \qquad \Delta;\Gamma\vdash\texttt{e}_0:\texttt{T}_0 \qquad \mathit{mtypemax}(\texttt{m},|\texttt{T}_0|_\Delta) = \bar{\texttt{C}}\to\texttt{D} \qquad \texttt{D} = |\texttt{T}|_\Delta}{|\texttt{e}_0\texttt{.m<}\bar{\texttt{V}}\texttt{>(}\bar{\texttt{e}}\texttt{)}|_{\Delta,\Gamma} = |\texttt{e}_0|_{\Delta,\Gamma}\texttt{.m(}|\bar{\texttt{e}}|_{\Delta,\Gamma}\texttt{)}} \qquad \text{(E-INVK)}$$

$$\frac{\Delta;\Gamma\vdash \texttt{e}_0\texttt{.m<}\bar{\texttt{V}}\texttt{>(}\bar{\texttt{e}}\texttt{)}:\texttt{T} \qquad \Delta;\Gamma\vdash\texttt{e}_0:\texttt{T}_0 \qquad \mathit{mtypemax}(\texttt{m},|\texttt{T}_0|_\Delta) = \bar{\texttt{C}}\to\texttt{D} \qquad \texttt{D} \neq |\texttt{T}|_\Delta}{|\texttt{e}_0\texttt{.m<}\bar{\texttt{V}}\texttt{>(}\bar{\texttt{e}}\texttt{)}|_{\Delta,\Gamma} = (|\texttt{T}|_\Delta)^s|\texttt{e}_0|_{\Delta,\Gamma}\texttt{.m(}|\bar{\texttt{e}}|_{\Delta,\Gamma}\texttt{)}} \qquad \text{(E-INVK-CAST)}$$

$$|\texttt{new N(}\bar{\texttt{e}}\texttt{)}|_{\Delta,\Gamma} = \texttt{new }|\texttt{N}|_\Delta\texttt{(}|\bar{\texttt{e}}|_{\Delta,\Gamma}\texttt{)} \qquad \text{(E-NEW)}$$

$$|\texttt{(N)e}_0|_{\Delta,\Gamma} = (|\texttt{N}|_\Delta)\ |\texttt{e}_0|_{\Delta,\Gamma} \qquad \text{(E-CAST)}$$

(Strictly speaking, we should think of the erasure operation as acting on typing derivations rather than expressions. Since well-typed expressions are in 1-1 correspondence with their typing derivations, the abuse of notation creates no confusion).

4.4 Erasure of Methods and Classes

The erasure of a method m with respect to type environment Δ in class C, written $|\mathtt{M}|_{\Delta,\mathtt{C}}$, is defined as follows:

$$\frac{\begin{array}{ll}\Gamma = \bar{\mathtt{x}}:\bar{\mathtt{T}}, \mathtt{this}:\mathtt{C}\texttt{<}\bar{\mathtt{X}}\texttt{>} & \Delta = \bar{\mathtt{X}} \texttt{<:} \bar{\mathtt{N}}, \bar{\mathtt{Y}}\texttt{<:}\bar{\mathtt{P}} \\ \mathit{mtypemax}(\mathtt{m},\mathtt{C}) = \bar{\mathtt{D}} \to \mathtt{D} & \mathtt{e}_i = \begin{cases} \mathtt{x}_i{}' & \text{if } \mathtt{D}_i = |\mathtt{T}_i|_\Delta \\ (|\mathtt{T}_i|_\Delta)^s \mathtt{x}_i{}' & \text{otherwise} \end{cases}\end{array}}{|\texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\texttt{>}\ \mathtt{T}\ \mathtt{m}(\bar{\mathtt{T}}\ \bar{\mathtt{x}})\texttt{\{}\ \mathtt{return}\ \mathtt{e}_0\texttt{;}\ \texttt{\}}|_{\bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}},\mathtt{C}} = \mathtt{D}\ \mathtt{m}(\bar{\mathtt{D}}\ \bar{\mathtt{x}}')\texttt{\{}\ \mathtt{return}\ [\bar{\mathtt{e}}/\bar{\mathtt{x}}]|\mathtt{e}_0|_{\Delta,\Gamma}\texttt{;}\ \texttt{\}}} \quad \text{(E-METHOD)}$$

The erasure of a method definition involves one subtlety, as discussed in the example of PairOfA. When the erasure $|\mathtt{T}_i|_\Delta$ of the type of a parameter is different from the corresponding argument type from *mtypemax*, the synthetic cast $(|\mathtt{T}_i|_\Delta)^s$ has to be inserted everywhere the parameter appears.

Remark. In GJ, the actual erasure is somewhat more complex, involving the introduction of bridge methods, so that one ends up with two overloaded methods: one with the maximum type and one with the instantiated type. For example, the erasure of PairOfA would be

```
class PairOfA extends Pair {
  PairOfA(Object fst, Object snd) {
    super(fst, snd);
  }
  Pair setfst(A newfst) {
    return new PairOfA(newfst, (A)this.snd);
  }
  Pair setfst(Object newfst) {
    return this.setfst((A)newfst);
  }
}
```

where the second definition of setfst is the bridge method, which overrides the definition of setfst in Pair. We do not model that extra complexity here, because it depends on overloading of method names, which is not modeled in FJ; here, instead, the rule E-METHOD merges two methods into one by inline-expanding the body of the actual method into the body of the bridge method.

The erasure of constructors and classes is

$$\begin{array}{l}|\mathtt{C}(\bar{\mathtt{U}}\ \bar{\mathtt{g}},\ \bar{\mathtt{T}}\ \bar{\mathtt{f}})\ \texttt{\{}\mathtt{super}(\bar{\mathtt{g}})\texttt{;}\ \mathtt{this}.\bar{\mathtt{f}}\ \texttt{=}\ \bar{\mathtt{f}}\texttt{;\}}|_\mathtt{C} \\ \quad = \mathtt{C}(\mathit{fieldsmax}(\mathtt{C}))\ \texttt{\{}\mathtt{super}(\bar{\mathtt{g}})\texttt{;}\ \mathtt{this}.\bar{\mathtt{f}}\ \texttt{=}\ \bar{\mathtt{f}}\texttt{;\}}\end{array} \quad \text{(E-CONSTRUCTOR)}$$

$$\frac{\Delta = \bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}}}{\begin{array}{l}|\mathtt{class}\ \mathtt{C}\texttt{<}\bar{\mathtt{X}}\ \mathtt{extends}\ \bar{\mathtt{N}}\texttt{>}\ \mathtt{extends}\ \mathtt{N}\ \texttt{\{}\bar{\mathtt{T}}\ \bar{\mathtt{f}}\texttt{;}\ \mathtt{K}\ \bar{\mathtt{M}}\texttt{\}}| \\ \quad = \mathtt{class}\ \mathtt{C}\ \mathtt{extends}\ |\mathtt{N}|_\Delta\texttt{\{}|\bar{\mathtt{T}}|_\Delta\ \bar{\mathtt{f}}\texttt{;}\ |\mathtt{K}|_\mathtt{C}\ |\bar{\mathtt{M}}|_{\Delta,\mathtt{C}}\texttt{\}}\end{array}} \quad \text{(E-CLASS)}$$

We write $|CT|$ for the erasure of a class table CT, defined in the obvious way.

$$\begin{tikzcd}
\mathtt{e} \arrow[r, "\text{reduce (FGJ)}"] \arrow[d, "\text{erase}"'] & \mathtt{e}' \arrow[d, dotted, "\text{erase}"] \\
|\mathtt{e}| \arrow[r, dotted, "\text{reduce (FJ)}"'] & |\mathtt{e}'|
\end{tikzcd}$$

Fig. 9. Commuting diagram.

4.5 Properties of Compilation

Having defined erasure, we may investigate some of its properties. As in the discussion of backward compatibility, we often use subscripts FJ or FGJ to avoid confusion.

Preservation of typing. First, a well-typed FGJ program erases to a well-typed FJ program, as expected; moreover, synthetic casts are not stupid.

THEOREM 4.5.1 (Erasure Preserves Typing). *If an FGJ class table CT is ok and* $\Delta;\Gamma \vdash_{\mathrm{FGJ}} \mathtt{e}:\mathtt{T}$, *then* $|CT|$ *is ok using the FJ typing rules and* $|\Gamma|_\Delta \vdash_{\mathrm{FJ}} |\mathtt{e}|_{\Delta,\Gamma} : |\mathtt{T}|_\Delta$. *Moreover, every synthetic cast in* $|CT|$ *and* $|\mathtt{e}|_{\Delta,\Gamma}$ *does not involve a stupid warning.*

PROOF. See Appendix A.3. □

Preservation of execution. More interestingly, we would intuitively expect that erasure from FGJ to FJ should also preserve the reduction behavior of FGJ programs, as in the commuting diagram shown in Figure 9. Unfortunately, this is not quite true. For example, consider the FGJ expression

$$\mathtt{e} = \mathtt{new\ Pair<A,B>(a,b).fst},$$

where `a` and `b` are expressions of type `A` and `B`, respectively, and consider its erasure

$$|\mathtt{e}|_{\Delta,\Gamma} = \mathtt{(A)}^s\mathtt{new\ Pair(}|\mathtt{a}|_{\Delta,\Gamma}\mathtt{,}|\mathtt{b}|_{\Delta,\Gamma}\mathtt{).fst}.$$

In FGJ, `e` reduces to `a`, while the erasure $|\mathtt{e}|_{\Delta,\Gamma}$ reduces to $\mathtt{(A)}^s|\mathtt{a}|_{\Delta,\Gamma}$ in FJ; it does not reduce to $|\mathtt{a}|_{\Delta,\Gamma}$ when `a` is not a `new` expression. (Note that it is not an artifact of our nondeterministic reduction strategy: it happens even if we adopt a call-by-value reduction strategy, since, after method invocation, we may obtain an expression like $\mathtt{(A)}^s\mathtt{e}$ where `e` is not a `new` expression.) Thus, the above diagram does not commute even if one-step reduction ($\longrightarrow$) at the bottom is replaced with many-step reduction ($\longrightarrow^*$). In general, synthetic casts can persist for a while in the FJ expression, although we expect those casts will eventually turn out to be upcasts when `a` reduces to a `new` expression.

In the example above, an FJ expression `d` reduced from $|\mathtt{e}|_{\Delta,\Gamma}$ had *more* synthetic casts than $|\mathtt{e}'|_{\Delta,\Gamma}$. However, this is not always the case: `d` may have *less* casts than $|\mathtt{e}'|_{\Delta,\Gamma}$ when the reduction step involves method invocation. Consider the FGJ expression

$$\mathtt{e} = \mathtt{new\ Pair<A,B>(a,\ b).setfst<B>(b')}$$

and its erasure

$$|\mathtt{e}|_{\Delta,\Gamma} = \mathtt{new\ Pair(}|\mathtt{a}|_{\Delta,\Gamma}\mathtt{,}|\mathtt{b}|_{\Delta,\Gamma}\mathtt{).setfst(}|\mathtt{b}'|_{\Delta,\Gamma}\mathtt{)}$$

where `a` is an expression of type `A` and `b` and $\mathtt{b}'$ are of type `B`. In FGJ,

$$\mathtt{e} \rightarrow_{\mathrm{FGJ}} \mathtt{new\ Pair\text{<}B,B\text{>}(b',new\ Pair\text{<}A,B\text{>}(a,b).snd)}.$$

In FJ, on the other hand,

$$|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\mathrm{FJ}} \mathtt{new\ Pair(}|\mathtt{b}'|_{\Delta,\Gamma}\mathtt{,new\ Pair(}|\mathtt{a}|_{\Delta,\Gamma}\mathtt{,}|\mathtt{b}|_{\Delta,\Gamma}\mathtt{).snd)}$$

which has fewer synthetic casts than

$$\mathtt{new\ Pair(}|\mathtt{b}'|_{\Delta,\Gamma}\mathtt{,(B)}^{s}\mathtt{new\ Pair(}|\mathtt{a}|_{\Delta,\Gamma}\mathtt{,}|\mathtt{b}|_{\Delta,\Gamma}\mathtt{).snd)},$$

which is the erasure of the reduced expression in FGJ. The subtlety we observe here is that when the erased term is reduced, synthetic casts may become "coarser" than the casts inserted when the reduced term is erased, or may be removed entirely as in this example. (Removal of downcasts can be considered as a combination of two operations: replacement of $\mathtt{(A)}^{s}$ with the coarser cast $\mathtt{(Object)}^{s}$ and removal of the upcast $\mathtt{(Object)}^{s}$, which does not affect the result of computation.)

To formalize both of these observations, we define an auxiliary relation that relates FJ expressions differing only by the addition and replacement of some synthetic casts. Suppose $\Gamma \vdash_{\mathrm{FJ}} \mathtt{e}:\mathtt{C}$. Let us call an expression `d` an *expansion* of `e` under Γ, written $\Gamma \vdash \mathtt{e} \overset{\mathrm{exp}}{\Rightarrow} \mathtt{d}$, if `d` is obtained from `e` by some combination of (1) addition of zero or more synthetic upcasts; (2) replacement of some synthetic casts $\mathtt{(D)}^{s}$ with $\mathtt{(C)}^{s}$, where `C` is a supertype of `D`; or (3) removal of some synthetic casts, and $\Gamma \vdash_{\mathrm{FJ}} \mathtt{d}:\mathtt{D}$ for some `D`.

Example 4.5.2. Suppose $\Gamma = \mathtt{x}:\mathtt{A}, \mathtt{y}:\mathtt{B}, \mathtt{z}:\mathtt{B}$ for given classes `A` and `B`. Then,

$$\Gamma \vdash \mathtt{x} \overset{\mathrm{exp}}{\Rightarrow} \mathtt{(A)}^{s}\mathtt{x}$$

and

$$\begin{aligned}\Gamma \vdash\ & \mathtt{new\ Pair(z,(B)}^{s}\mathtt{new\ Pair(x,y).snd)}\\ & \overset{\mathrm{exp}}{\Rightarrow} \mathtt{new\ Pair(z,new\ Pair(x,y).snd)}.\end{aligned}$$

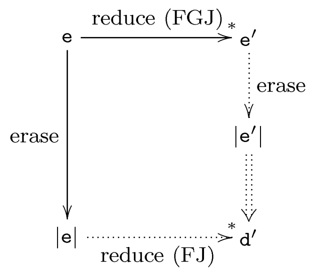
Then, reduction commutes with erasure modulo expansion:

THEOREM 4.5.3 (Erasure Preserves Reduction Modulo Expansion). *If* $\Delta;\Gamma \vdash \mathtt{e}:\mathtt{T}$ *and* $\mathtt{e} \rightarrow^{*}_{\mathrm{FGJ}} \mathtt{e}'$*, then there exists some FJ expression* $\mathtt{d}'$ *such that* $|\Gamma|_{\Delta} \vdash |\mathtt{e}'|_{\Delta,\Gamma} \overset{\mathrm{exp}}{\Rightarrow} \mathtt{d}'$ *and* $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow^{*}_{\mathrm{FJ}} \mathtt{d}'$. *In other words, the diagram in Figure* 10 *commutes.*

PROOF. See Appendix A.4. □

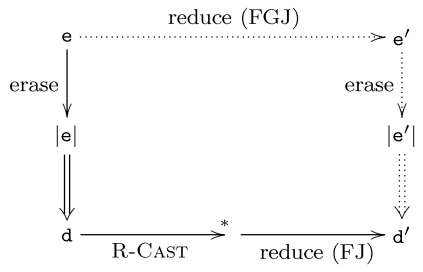
Conversely, for the execution of an erased expression, there is a corresponding execution in FGJ semantics:

THEOREM 4.5.4 (Erased Program Reflects FGJ Execution). *Suppose that* $\Delta;\Gamma \vdash \mathtt{e}:\mathtt{T}$ *and* $|\Gamma|_{\Delta} \vdash |\mathtt{e}|_{\Delta,\Gamma} \overset{\mathrm{exp}}{\Rightarrow} \mathtt{d}$. *If* `d` *reduces to* $\mathtt{d}'$ *with zero or more steps*



$$\begin{tikzcd}
\mathtt{e} \arrow[r, "\text{reduce (FGJ)}^{*}"] \arrow[dd, "\text{erase}"'] & \mathtt{e}' \arrow[d, dotted, "\text{erase}"] \\
 & |\mathtt{e}'| \arrow[d, dotted, Rightarrow] \\
|\mathtt{e}| \arrow[r, dotted, "*", "\text{reduce (FJ)}"'] & \mathtt{d}'
\end{tikzcd}$$

Fig. 10.



$$\begin{tikzcd}
\mathtt{e} \arrow[rr, dotted, "\text{reduce (FGJ)}"] \arrow[d, "\text{erase}"'] & & \mathtt{e}' \arrow[d, dotted, "\text{erase}"'] \\
|\mathtt{e}| \arrow[d, Rightarrow] & & |\mathtt{e}'| \arrow[d, dotted, Rightarrow] \\
\mathtt{d} \arrow[r, "*", "\text{R-Cast}"'] & {} \arrow[r, "\text{reduce (FJ)}"'] & \mathtt{d}'
\end{tikzcd}$$

Fig. 11.

by removing synthetic casts, followed by one step by other kinds of reduction, then $\mathtt{e} \rightarrow_{\mathrm{FGJ}} \mathtt{e}'$ *for some* $\mathtt{e}'$ *and* $|\Gamma|_{\Delta} \vdash |\mathtt{e}'|_{\Delta,\Gamma} \overset{\exp}{\Rightarrow} \mathtt{d}'$. *In other words, the diagram shown in Figure* 11 *commutes.*

PROOF. Also see Appendix A.4. □

As easy corollaries of these theorems, it can be shown that, if an FGJ expression e reduces to a "fully evaluated expression," then the erasure of e reduces to exactly its erasure and vice versa. Similarly, if FGJ reduction gets stuck at a stupid cast, then FJ reduction also gets stuck because of the same typecast and vice versa.

COROLLARY 4.5.5 (Erasure Preserves Execution Results). *If* $\Delta;\Gamma \vdash \mathtt{e}:\mathtt{T}$ *and* $\mathtt{e} \rightarrow_{\mathrm{FGJ}}^{*} \mathtt{w}$, *then* $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\mathrm{FJ}}{}^{*} |\mathtt{w}|_{\Delta,\Gamma}$. *Similarly, if* $\Delta;\Gamma \vdash \mathtt{e}:\mathtt{T}$ *and* $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\mathrm{FJ}}{}^{*} \mathtt{v}$, *then there exists an FGJ value* w *such that* $\mathtt{e} \rightarrow_{\mathrm{FGJ}}^{*} \mathtt{w}$ *and* $|\mathtt{w}|_{\Delta,\Gamma} = \mathtt{v}$.

PROOF. By Theorem 4.5.3, there must exist an FJ expression d such that $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\mathrm{FGJ}}^{*} \mathtt{d}$ and $|\Gamma|_{\Delta} \vdash |\mathtt{w}|_{\Delta,\Gamma} \overset{\exp}{\Rightarrow} \mathtt{d}$. Since the FJ value $|\mathtt{w}|_{\Delta,\Gamma}$ does not include any typecasts, d is obtained only by adding some (synthetic) upcasts. Therefore, d reduces to $|\mathtt{w}|_{\Delta,\Gamma}$.

The second part follows from a similar argument using Theorem 4.5.4. □

COROLLARY 4.5.6 (Erasure Preserves Typecast Errors). *If* $\Delta;\Gamma \vdash \mathtt{e}:\mathtt{T}$ *and* $\mathtt{e} \rightarrow_{\mathrm{FGJ}}^{*} \mathtt{e}'$, *where* $\mathtt{e}'$ *has a stuck subexpression* $\mathtt{(C<\bar{S}>)new\ D<\bar{T}>(\bar{e})}$, *then* $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\mathrm{FJ}}{}^{*} \mathtt{d}'$ *such that* $\mathtt{d}'$ *has a stuck subexpression* $\mathtt{(C)new\ D(\bar{d})}$, *where* $\bar{\mathtt{d}}$ *are expansions of the erasures of* $\bar{\mathtt{e}}$, *at the same position (modulo synthetic casts) as the erasure of* $\mathtt{e}'$. *Similarly, if* $\Delta;\Gamma \vdash \mathtt{e}:\mathtt{T}$ *and* $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\mathrm{FJ}}{}^{*} \mathtt{e}'$, *where* $\mathtt{e}'$ *has a stuck subexpression* $\mathtt{(C)new\ D(\bar{e})}$, *then there exists an FGJ expression*

P1: IBD
CM026A-03 ACM-TRANSACTION January 23, 2002 17:39

d *such that* $\mathtt{e} \rightarrow_{\mathrm{FGJ}}^{*} \mathtt{d}$ *and* $|\Gamma|_{\Delta} \vdash |\mathtt{d}|_{\Delta,\Gamma} \overset{\mathrm{exp}}{\Rightarrow} \mathtt{e}'$ *and* d *has a stuck subexpression* $\mathtt{(C<\bar{S}>)new\ D<\bar{T}>(\bar{d})}$, *where* $\bar{\mathtt{e}}$ *are expansions of the erasures of* $\bar{\mathtt{d}}$, *at the same position (modulo synthetic casts) as* $\mathtt{e}'$.

PROOF. Similar to the proof of Corollary 4.5.5 using Theorem 4.5.4. □

5. RELATED WORK

Core calculi for Java. There are several known proofs in the literature of type soundness for subsets of Java. In the earliest, Drossopoulou et al. [1999] (using a technique later mechanically checked by Syme [1997]) prove soundness for a fairly large subset of sequential Java. Like us, they use a small-step operational semantics, but they avoid the subtleties of "stupid casts" by omitting casting entirely. Nipkow and von Oheimb [1998] give a mechanically checked proof of soundness for a somewhat larger core language. Their language does include casts, but it is formulated using a "big-step" operational semantics, which sidesteps the stupid cast problem. Flatt et al. [1998a; 1998b] use a small-step semantics and formalize a language with both assignment and casting. Their system is somewhat larger than ours (the syntax, typing, and operational semantics rules take perhaps three times the space), and the soundness proof, though correspondingly longer, is of similar complexity. Their published proof of subject reduction in the earlier version is slightly flawed—the case that motivated our introduction of stupid casts is not handled properly—but the problem can be repaired by applying the same refinement we have used here.

Of these three studies, that of Flatt et al. is closest to ours in an important sense: the goal there, as here, is to choose a core calculus that is as *small* as possible, capturing just the features of Java that are relevant to some particular task. In their case, the task is analyzing an extension of Java with Common Lisp style mixins—in ours, extensions of the core type system. The goal of the other two systems, on the other hand, is to include as *large* a subset of Java as possible, since their primary interest is proving the soundness of Java itself.

Other class-based object calculi. The literature on foundations of object-oriented languages contains many papers formalizing class-based object-oriented languages, either taking classes as primitive (e.g., Wand [1989], Bruce [1994], Bono et al. [1999a; 1999b]) or translating classes into lower-level mechanisms (e.g., Fisher and Mitchell [1998], Bono and Fisher [1998], Abadi and Cardelli [1996], and Pierce and Turner [1994]). Some of these systems (e.g., Pierce and Turner [1994] and Bruce [1994]) include generic classes and methods, but only in fairly simple forms.

Generic extensions of Java. A number of extensions of Java with generic classes and methods have been proposed by various groups, including the language of Agesen et al. [1997]; PolyJ, by Myers et al. [1997]; Pizza, by Odersky and Wadler [1997]; GJ, by Bracha et al. [1998]; NextGen, by Cartwright and Steele Jr. [1998]; and LM, by Viroli and Natali [2000]. While all these languages are believed to be typesafe, our study of FGJ is the first to give rigorous proof of soundness for a generic extension of Java. We have used GJ as the basis for our generic extension, but similar techniques should apply to the forms of genericity found in the rest of these languages.

Recently, Duggan [1999] has proposed a technique to translate monomorphic classes to parametric classes by inferring type argument information. He has also defined a polymorphic extension of Java, slightly less expressive than GJ (for example, polymorphic methods are not allowed, and a subclass must have the same number of type arguments as its superclass). The type soundness theorem of the language is mentioned, but the stupid cast problem is not taken into account.

6. DISCUSSION

We have presented Featherweight Java, a core language for Java modeled closely on the lambda-calculus and embodying many of the key features of Java's type system. FJ's definition and proof of soundness are both concise and straightforward, making it a suitable arena for the study of ambitious extensions to the type system, such as the generic types of GJ. We have developed this extension in detail, stated some of its fundamental properties, and given their proofs.

It was pleasing to discover that FGJ could be formulated as a straightforward extension of FJ, giving us additional confidence that the design of GJ was on the right track. Our investigation of FGJ led us to uncover one bug in the compiler, involving a subtle relation between subtyping and raw types (see below). Most importantly, however, FGJ has given us useful vocabulary and notation for thinking about the design of GJ.

FJ itself is not quite complete enough to model some of the interesting subtleties found in GJ. In particular, the full GJ language allows some parameters to be instantiated by a special "bottom type" `*`, using a delicate rule to avoid unsoundness in the presence of assignment. Moreover, nonstandard subtyping like `C<*>` <: `C<T>` is allowed when the type argument of the left-hand side is `*` (recall that type constructors are invariant). Capturing the relevant issues in FGJ would require extending it with assignment and `null` values (both of these extensions seem straightforward, but cost us some of the pleasing compactness of FJ as it stands). Another subtle aspect of GJ that is not accurately modeled in FGJ is the use of bridge methods in the compilation from GJ to JVM bytecodes. To treat this compilation exactly as GJ does, we would need to extend FJ with overloading.

The present formalization of GJ also does not include *raw types*, a unique aspect of the GJ design that supports compatibility between old, unparameterized code and new, parameterized code. We are currently experimenting with an extension of FGJ with raw types. A preliminary result [Igarashi et al. 2001] has already uncovered that the currently implemented typing system (version 0.6m, as of August 1999) of raw types is unsound; a repaired version of the type system to be incorporated in the next release is proved to be sound.

Formalizing generics has proven to be a useful application domain for FJ, but there are other areas where its extreme simplicity may yield significant leverage. Igarashi and Pierce [2000] formalized a core of Java 1.1's *inner classes* on top of FJ; League, et al. [2001] have developed type-preserving compilation of FJ to a typed intermediate language; Studer [2000] studied a recursion-theoretic denotational semantics of FJ; Schultz [2001] has used a variant of FJ as a formal basis of partial evaluation for class-based object-oriented languages; and Ancona and Zucca [2001] have developed a module language for Java, where its core language used for formalization is very close to FJ.

APPENDIX

A.1 Proof of Theorem 2.4.1

Before giving the proof, we develop a number of required lemmas.

LEMMA A.1.1. *If* $mtype(\mathtt{m},\mathtt{D}) = \bar{\mathtt{C}} \to \mathtt{C}_0$, *then* $mtype(\mathtt{m},\mathtt{C}) = \bar{\mathtt{C}} \to \mathtt{C}_0$ *for all* $\mathtt{C} <: \mathtt{D}$.

PROOF. Straightforward induction on the derivation of $\mathtt{C} <: \mathtt{D}$. Note that whether `m` is defined in $CT(\mathtt{C})$ or not, $mtype(\mathtt{m},\mathtt{C})$ should be the same as $mtype(\mathtt{m},\mathtt{E})$ where `class` $\mathtt{C} \triangleleft \mathtt{E}$ `{...}`. □

LEMMA A.1.2 (Term Substitution Preserves Typing). *If* $\Gamma, \bar{\mathtt{x}} : \bar{\mathtt{B}} \vdash \mathtt{e} : \mathtt{D}$, *and* $\Gamma \vdash \bar{\mathtt{d}} : \bar{\mathtt{A}}$ *where* $\bar{\mathtt{A}} <: \bar{\mathtt{B}}$, *then* $\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e} : \mathtt{C}$ *for some* $\mathtt{C} <: \mathtt{D}$.

PROOF. By induction on the derivation of $\Gamma, \bar{\mathtt{x}} : \bar{\mathtt{B}} \vdash \mathtt{e} : \mathtt{D}$. The intuitions are exactly the same as for the lambda-calculus with subtyping (details vary a little, of course).

Case T-VAR. $\mathtt{e} = \mathtt{x} \qquad \mathtt{D} = \Gamma(\mathtt{x})$

If $\mathtt{x} \notin \bar{\mathtt{x}}$, then the conclusion is immediate, since $[\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{x} = \mathtt{x}$. On the other hand, if $\mathtt{x} = \mathtt{x}_i$ and $\mathtt{D} = \mathtt{B}_i$, then, since $[\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{x} = [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{x}_i = \mathtt{d}_i$, letting $\mathtt{C} = \mathtt{A}_i$ finishes the case.

Case T-FIELD. $\mathtt{e} = \mathtt{e}_0.\mathtt{f}_i \qquad \Gamma, \bar{\mathtt{x}} : \bar{\mathtt{B}} \vdash \mathtt{e}_0 : \mathtt{D}_0$
$fields(\mathtt{D}_0) = \bar{\mathtt{C}}\ \bar{\mathtt{f}} \qquad \mathtt{D} = \mathtt{C}_i$

By the induction hypothesis, there is some $\mathtt{C}_0$ such that $\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0 : \mathtt{C}_0$ and $\mathtt{C}_0 <: \mathtt{D}_0$. Then, it is easy to show that

$$fields(\mathtt{C}_0) = fields(\mathtt{D}_0), \bar{\mathtt{D}}\ \bar{\mathtt{g}}$$

for some $\bar{\mathtt{D}}\ \bar{\mathtt{g}}$. Therefore, by the rule T-FIELD, $\Gamma \vdash ([\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0).\mathtt{f}_i : \mathtt{C}_i$.

Case T-INVK. $\mathtt{e} = \mathtt{e}_0.\mathtt{m}(\bar{\mathtt{e}}) \qquad \Gamma, \bar{\mathtt{x}} : \bar{\mathtt{B}} \vdash \mathtt{e}_0 : \mathtt{D}_0 \qquad mtype(\mathtt{m},\mathtt{D}_0) = \bar{\mathtt{E}} \to \mathtt{D}$
$\Gamma, \bar{\mathtt{x}} : \bar{\mathtt{B}} \vdash \bar{\mathtt{e}} : \bar{\mathtt{D}} \qquad \bar{\mathtt{D}} <: \bar{\mathtt{E}}$

By the induction hypothesis, there are some $\mathtt{C}_0$ and $\bar{\mathtt{C}}$ such that

$$\begin{array}{ll} \Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0 : \mathtt{C}_0 & \mathtt{C}_0 <: \mathtt{D}_0 \\ \Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\bar{\mathtt{e}} : \bar{\mathtt{C}} & \bar{\mathtt{C}} <: \bar{\mathtt{D}} \end{array}$$

By Lemma A.1.1, $mtype(\mathtt{m},\mathtt{C}_0) = \bar{\mathtt{E}} \to \mathtt{D}$. Then, $\bar{\mathtt{C}} <: \bar{\mathtt{E}}$ by the transitivity of $<:$. Therefore, by the rule T-INVK, $\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0.\mathtt{m}([\bar{\mathtt{d}}/\bar{\mathtt{x}}]\bar{\mathtt{e}}) : \mathtt{D}$.

Case T-NEW. $\mathtt{e} = \mathtt{new}\ \mathtt{D}(\bar{\mathtt{e}}) \qquad fields(\mathtt{D}) = \bar{\mathtt{D}}\ \bar{\mathtt{f}}$
$\Gamma, \bar{\mathtt{x}} : \bar{\mathtt{B}} \vdash \bar{\mathtt{e}} : \bar{\mathtt{C}} \qquad \bar{\mathtt{C}} <: \bar{\mathtt{D}}$

By the induction hypothesis, there are $\bar{\mathtt{E}}$ such that $\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\bar{\mathtt{e}} : \bar{\mathtt{E}}$ and $\bar{\mathtt{E}} <: \bar{\mathtt{C}}$. Then, $\bar{\mathtt{E}} <: \bar{\mathtt{D}}$, by transitivity of $<:$. Therefore, by the rule T-New, $\Gamma \vdash \mathtt{new\ D}([\bar{\mathtt{d}}/\bar{\mathtt{x}}]\bar{\mathtt{e}}) : \mathtt{D}$.

Case T-UCast. $\mathtt{e} = (\mathtt{D})\mathtt{e}_0 \quad \Gamma, \bar{\mathtt{x}} : \bar{\mathtt{B}} \vdash \mathtt{e}_0 : \mathtt{C} \quad \mathtt{C} <: \mathtt{D}$

By the induction hypothesis, there is some $\mathtt{E}$ such that $\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0 : \mathtt{E}$ and $\mathtt{E} <: \mathtt{C}$. Then, $\mathtt{E} <: \mathtt{D}$ by transitivity of $<:$; this yields $\Gamma \vdash (\mathtt{D})([\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0) : \mathtt{D}$ by the rule T-UCast.

Case T-DCast. $\mathtt{e} = (\mathtt{D})\mathtt{e}_0 \quad \Gamma, \bar{\mathtt{x}} : \bar{\mathtt{B}} \vdash \mathtt{e}_0 : \mathtt{C} \quad \mathtt{D} <: \mathtt{C} \quad \mathtt{D} \neq \mathtt{C}$

By the induction hypothesis, there is some $\mathtt{E}$ such that $\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0 : \mathtt{E}$ and $\mathtt{E} <: \mathtt{C}$. If $\mathtt{E} <: \mathtt{D}$ or $\mathtt{D} <: \mathtt{E}$, then $\Gamma \vdash (\mathtt{D})([\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0) : \mathtt{D}$ by the rule T-UCast or T-DCast, respectively. On the other hand, if both $\mathtt{D} \not<: \mathtt{E}$ and $\mathtt{E} \not<: \mathtt{D}$, then $\Gamma \vdash (\mathtt{D})([\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0) : \mathtt{D}$ (with a *stupid warning*) by the rule T-SCast.

Case T-SCast. $\mathtt{e} = (\mathtt{D})\mathtt{e}_0 \quad \Gamma, \bar{\mathtt{x}} : \bar{\mathtt{B}} \vdash \mathtt{e}_0 : \mathtt{C} \quad \mathtt{D} \not<: \mathtt{C} \quad \mathtt{C} \not<: \mathtt{D}$

By the induction hypothesis, there is some $\mathtt{E}$ such that $\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0 : \mathtt{E}$ and $\mathtt{E} <: \mathtt{C}$. This means that $\mathtt{E} \not<: \mathtt{D}$. (To see this, note that each class in FJ has just one superclass. It follows that if both $\mathtt{E} <: \mathtt{C}$ and $\mathtt{E} <: \mathtt{D}$, then either $\mathtt{C} <: \mathtt{D}$ or $\mathtt{D} <: \mathtt{C}$). So $\Gamma \vdash (\mathtt{D})([\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0) : \mathtt{D}$ (with a *stupid warning*), by T-SCast. □

Lemma A.1.3 (Weakening). *If* $\Gamma \vdash \mathtt{e} : \mathtt{C}$, *then* $\Gamma, \mathtt{x} : \mathtt{D} \vdash \mathtt{e} : \mathtt{C}$.

Proof. Straightforward induction. □

Lemma A.1.4. *If* $mtype(\mathtt{m}, \mathtt{C}_0) = \bar{\mathtt{D}} \to \mathtt{D}$, *and* $mbody(\mathtt{m}, \mathtt{C}_0) = \bar{\mathtt{x}}.\mathtt{e}$, *then, for some* $\mathtt{D}_0$ *with* $\mathtt{C}_0 <: \mathtt{D}_0$, *there exists* $\mathtt{C} <: \mathtt{D}$ *such that* $\bar{\mathtt{x}} : \bar{\mathtt{D}}, \mathtt{this} : \mathtt{D}_0 \vdash \mathtt{e} : \mathtt{C}$.

Proof. By induction on the derivation of $mbody(\mathtt{m}, \mathtt{C}_0)$. The base case (where $\mathtt{m}$ is defined in $\mathtt{C}_0$) is easy, since $\mathtt{m}$ is defined in $CT(\mathtt{C}_0)$ and $\bar{\mathtt{x}} : \bar{\mathtt{D}}, \mathtt{this} : \mathtt{C}_0 \vdash \mathtt{e} : \mathtt{C}$ by the T-Method. The induction step is also straightforward. □

We are now ready to give the proof of the subject reduction theorem.

Proof of Theorem 2.4.1. By induction on a derivation of $\mathtt{e} \to \mathtt{e}'$, with a case analysis on the reduction rule used.

Case R-Field. $\mathtt{e} = (\mathtt{new\ C}_0(\bar{\mathtt{e}})).\mathtt{f}_i \quad \mathtt{e}' = \mathtt{e}_i \quad \mathit{fields}(\mathtt{C}_0) = \bar{\mathtt{D}}\ \bar{\mathtt{f}}$

By rule T-Field, we have

$$\Gamma \vdash \mathtt{new\ C}_0(\bar{\mathtt{e}}) : \mathtt{D}_0 \quad \mathtt{C} = \mathtt{D}_i$$

for some $\mathtt{D}_0$. Again, by the rule T-New,

$$\Gamma \vdash \bar{\mathtt{e}} : \bar{\mathtt{C}} \quad \bar{\mathtt{C}} <: \bar{\mathtt{D}} \quad \mathtt{D}_0 = \mathtt{C}_0$$

In particular, $\Gamma \vdash \mathtt{e}_i : \mathtt{C}_i$, finishing the case, since $\mathtt{C}_i <: \mathtt{D}_i$.

Case R-Invk. $\mathtt{e} = (\mathtt{new\ C}_0(\bar{\mathtt{e}})).\mathtt{m}(\bar{\mathtt{d}}) \qquad mbody(\mathtt{m}, \mathtt{C}_0) = \bar{\mathtt{x}}.\mathtt{e}_0$
$\mathtt{e}' = [\bar{\mathtt{d}}/\bar{\mathtt{x}}, \mathtt{new\ C}_0(\bar{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0$

By the rules T-Invk and T-New, we have

$$\begin{array}{ll} \Gamma \vdash \mathtt{new\ C}_0(\bar{\mathtt{e}}) : \mathtt{C}_0 & mtype(\mathtt{m}, \mathtt{C}_0) = \bar{\mathtt{D}} \to \mathtt{C} \\ \Gamma \vdash \bar{\mathtt{d}} : \bar{\mathtt{C}} & \bar{\mathtt{C}} <: \bar{\mathtt{D}} \end{array}$$

for some $\bar{\texttt{C}}$ and $\bar{\texttt{D}}$. By Lemma A.1.4, $\bar{\texttt{x}} : \bar{\texttt{D}}, \texttt{this} : \texttt{D}_0 \vdash \texttt{e}_0 : \texttt{B}$ for some $\texttt{D}_0$ and $\texttt{B}$ where $\texttt{C}_0 \mathrel{<:} \texttt{D}_0$ and $\texttt{B} \mathrel{<:} \texttt{C}$. By Lemma A.1.3, $\Gamma, \bar{\texttt{x}} : \bar{\texttt{D}}, \texttt{this} : \texttt{D}_0 \vdash \texttt{e}_0 : \texttt{B}$. Then, by Lemma A.1.2, $\Gamma \vdash [\bar{\texttt{d}}/\bar{\texttt{x}}, \texttt{new C}_0\texttt{(}\bar{\texttt{e}}\texttt{)/this}]\texttt{e}_0 : \texttt{E}$ for some $\texttt{E} \mathrel{<:} \texttt{B}$. Then $\texttt{E} \mathrel{<:} \texttt{C}$ by transitivity of $<:$. Finally, letting $\texttt{C}' = \texttt{E}$ finishes this case.

Case R-Cast. $\texttt{e} = \texttt{(D)(new C}_0\texttt{(}\bar{\texttt{e}}\texttt{))}$ $\texttt{C}_0 \mathrel{<:} \texttt{D}$ $\texttt{e}' = \texttt{new C}_0\texttt{(}\bar{\texttt{e}}\texttt{)}$

The proof of $\Gamma \vdash \texttt{(D)(new C}_0\texttt{(}\bar{\texttt{e}}\texttt{))} : \texttt{C}$ must end with the rule T-UCast, since the derivation ending with T-SCast or T-DCast contradicts the assumption $\texttt{C}_0 \mathrel{<:} \texttt{D}$. By the rules T-UCast and T-New, we have $\Gamma \vdash \texttt{new C}_0\texttt{(}\bar{\texttt{e}}\texttt{)} : \texttt{C}_0$ and $\texttt{D} = \texttt{C}$, which finish the case.

The cases for congruence rules are easy. We show just one:

Case RC-Cast. $\texttt{e} = \texttt{(D)e}_0$ $\texttt{e}' = \texttt{(D)e}_0{}'$ $\texttt{e}_0 \to \texttt{e}_0{}'$

There are three subcases, according to the last typing rule used.

Subcase T-UCast. $\Gamma \vdash \texttt{e}_0 : \texttt{C}_0$ $\texttt{C}_0 \mathrel{<:} \texttt{D}$ $\texttt{D} = \texttt{C}$

By the induction hypothesis, $\Gamma \vdash \texttt{e}_0{}' : \texttt{C}_0{}'$ for some $\texttt{C}_0{}' \mathrel{<:} \texttt{C}_0$. Then, $\texttt{C}_0{}' \mathrel{<:} \texttt{C}$, by transitivity of $<:$. Therefore, by the rule T-UCast, $\Gamma \vdash \texttt{(C)e}_0{}' : \texttt{C}$ (without any additional *stupid warning*).

Subcase T-DCast. $\Gamma \vdash \texttt{e}_0 : \texttt{C}_0$ $\texttt{D} \mathrel{<:} \texttt{C}_0$ $\texttt{D} = \texttt{C} \neq \texttt{C}_0$

By the induction hypothesis, $\Gamma \vdash \texttt{e}_0{}' : \texttt{C}_0{}'$ for some $\texttt{C}_0{}' \mathrel{<:} \texttt{C}_0$. If either $\texttt{C}_0{}' \mathrel{<:} \texttt{C}$ or $\texttt{C} \mathrel{<:} \texttt{C}_0{}'$, then $\Gamma \vdash \texttt{(C)e}_0{}' : \texttt{C}$ by the rule T-UCast or T-DCast (without any additional *stupid warning*). On the other hand, if both $\texttt{C}_0{}' \not\mathrel{<:} \texttt{C}$ and $\texttt{C} \not\mathrel{<:} \texttt{C}_0{}'$, then, $\Gamma \vdash \texttt{(C)e}_0{}' : \texttt{C}$ with *stupid warning* by the rule T-SCast.

Subcase T-SCast. $\Gamma \vdash \texttt{e}_0 : \texttt{C}_0$ $\texttt{D} \not\mathrel{<:} \texttt{C}_0$ $\texttt{C}_0 \not\mathrel{<:} \texttt{D}$ $\texttt{D} = \texttt{C}$

By the induction hypothesis, $\Gamma \vdash \texttt{e}_0{}' : \texttt{C}_0{}'$ for some $\texttt{C}_0{}' \mathrel{<:} \texttt{C}_0$. Then, both $\texttt{C}_0{}' \not\mathrel{<:} \texttt{C}$ and $\texttt{C} \not\mathrel{<:} \texttt{C}_0{}'$ also hold, following the same argument found in the proof of Lemma A.1.2 (the case for T-SCast). Therefore, $\Gamma \vdash \texttt{(C)e}_0{}' : \texttt{C}$ with *stupid warning*.

A.2 Proof of Theorem 3.4.1

Before giving the proof, we develop a number of required lemmas.

Lemma A.2.1 (Weakening). *Suppose* $\Delta, \bar{\texttt{X}} \mathrel{<:} \bar{\texttt{N}} \vdash \bar{\texttt{N}}$ ok *and* $\Delta \vdash \texttt{U}$ ok.

(1) *If* $\Delta \vdash \texttt{S} \mathrel{<:} \texttt{T}$, *then* $\Delta, \bar{\texttt{X}} \mathrel{<:} \bar{\texttt{N}} \vdash \texttt{S} \mathrel{<:} \texttt{T}$.
(2) *If* $\Delta \vdash \texttt{S}$ ok, *then* $\Delta, \bar{\texttt{X}} \mathrel{<:} \bar{\texttt{N}} \vdash \texttt{S}$ ok.
(3) *If* $\Delta; \Gamma \vdash \texttt{e} : \texttt{T}$, *then* $\Delta; \Gamma, \texttt{x} : \texttt{U} \vdash \texttt{e} : \texttt{T}$ *and* $\Delta, \bar{\texttt{X}} \mathrel{<:} \bar{\texttt{N}}; \Gamma \vdash \texttt{e} : \texttt{T}$.

Proof. Each of them is proved by straightforward induction on the derivation of $\Delta \vdash \texttt{S} \mathrel{<:} \texttt{T}$ and $\Delta \vdash \texttt{S}$ ok and $\Delta; \Gamma \vdash \texttt{e} : \texttt{T}$. □

Lemma A.2.2. *If* $\Delta \vdash \texttt{E<}\bar{\texttt{V}}\texttt{>} \mathrel{<:} \texttt{D<}\bar{\texttt{U}}\texttt{>}$ *and* $\texttt{D} \not\trianglelefteq \texttt{C}$ *and* $\texttt{C} \not\trianglelefteq \texttt{D}$, *then* $\texttt{E} \not\trianglelefteq \texttt{C}$ *and* $\texttt{C} \not\trianglelefteq \texttt{E}$.

Proof. It is easy to see that $\Delta \vdash \texttt{E<}\bar{\texttt{V}}\texttt{>} \mathrel{<:} \texttt{D<}\bar{\texttt{U}}\texttt{>}$ implies $\texttt{E} \trianglelefteq \texttt{D}$. The conclusions are easily proved by contradiction. (A similar argument is found in the proof of Lemma A.1.2.) □

LEMMA A.2.3. *Suppose dcast*(C, D) *and* $\Delta \vdash$ C<$\bar{T}$> <: D<$\bar{U}$>. *If* $\Delta \vdash$ C<$\bar{T}'$> <: D<$\bar{U}$>, *then* $\bar{T}' = \bar{T}$.

PROOF. The case where *dcast*(C, D) because *dcast*(C, E) and *dcast*(E, D) is easy: Note that from every derivation of $\Delta \vdash$ C<$\bar{T}$> <: D<$\bar{U}$> we can also derive $\Delta \vdash$ C<$\bar{T}$> <: E<$\bar{V}$> and $\Delta \vdash$ E<$\bar{V}$> <: D<$\bar{U}$> for some $\bar{V}$. Finally, if D is the direct superclass of C, by the rule S-CLASS, D<$\bar{U}$> = [$\bar{T}$/$\bar{X}$]D<$\bar{V}$> where class C<$\bar{X} \triangleleft \bar{N}$> $\triangleleft$ D<$\bar{V}$> {...} for some $\bar{V}$. Similarly, D<$\bar{U}$> = [$\bar{T}'$/$\bar{X}$]D<$\bar{V}$>, since *FV*($\bar{V}$) = $\bar{X}$. Then, it must be the case that $\bar{T} = \bar{T}'$, finishing the proof. □

LEMMA A.2.4 *If dcast*(C, E) *and* C $\trianglelefteq$ D $\trianglelefteq$ E *with* C $\neq$ D $\neq$ E, *then dcast*(C, D) *and dcast*(D, E).

PROOF. Easy. □

LEMMA A.2.5 (Type Substitution Preserves Subtyping). *If* Δ_1, $\bar{X}$ <: $\bar{N}$, $\Delta_2 \vdash$ S <: T *and* $\Delta_1 \vdash \bar{U}$ <: [$\bar{U}$/$\bar{X}$]$\bar{N}$ *with* $\Delta_1 \vdash \bar{U}$ ok *and none of* $\bar{X}$ *appearing in* Δ_1, *then* Δ_1, [$\bar{U}$/$\bar{X}$]$\Delta_2 \vdash$ [$\bar{U}$/$\bar{X}$]S <: [$\bar{U}$/$\bar{X}$]T.

PROOF. By induction on the derivation of Δ_1, $\bar{X}$ <: $\bar{N}$, $\Delta_2 \vdash$ S <: T.

Case S-REFL. Trivial.
Case S-TRANS, S-CLASS. Easy.
Case S-VAR. S = X T = (Δ_1, $\bar{X}$ <: $\bar{N}$, Δ_2)(X)

If X $\in$ *dom*(Δ_1) $\cup$ *dom*(Δ_2), then the conclusion is immediate. On the other hand, if X = X_i, then, by assumption, we have $\Delta_1 \vdash U_i$ <: [$\bar{U}$/$\bar{X}$]N_i. Finally, Lemma A.2.1 finishes the case. □

LEMMA A.2.6 (Type Substitution Preserves Type Well-Formedness). *If* Δ_1, $\bar{X}$ <: $\bar{N}$, $\Delta_2 \vdash$ T ok *and* $\Delta_1 \vdash \bar{U}$ <: [$\bar{U}$/$\bar{X}$]$\bar{N}$ *with* $\Delta_1 \vdash \bar{U}$ ok *and none of* $\bar{X}$ *appearing in* Δ_1, *then* Δ_1, [$\bar{U}$/$\bar{X}$]$\Delta_2 \vdash$ [$\bar{U}$/$\bar{X}$]T ok.

PROOF. By induction on the derivation of Δ_1, $\bar{X}$ <: $\bar{N}$, $\Delta_2 \vdash$ T ok, with a case analysis on the last rule used.

Case WF-OBJECT. Trivial.
Case WF-VAR. T = X X $\in$ *dom*(Δ_1, $\bar{X}$<:$\bar{N}$, Δ_2)

The case X $\in$ X_i follows from $\Delta_1 \vdash \bar{U}$ ok and Lemma A.2.1; otherwise immediate.

Case WF-CLASS. T = C<$\bar{T}$> Δ_1, $\bar{X}$<:$\bar{N}$, $\Delta_2 \vdash \bar{T}$ ok
Δ_1, $\bar{X}$ <: $\bar{N}$, $\Delta_2 \vdash \bar{T}$ <: [$\bar{T}$/$\bar{Y}$]$\bar{P}$
class C<$\bar{Y} \triangleleft \bar{P}$> $\triangleleft$ N {...}

By the induction hypothesis,

$$\Delta_1, [\bar{U}/\bar{X}]\Delta_2 \vdash [\bar{U}/\bar{X}]\bar{T} \text{ ok.}$$

On the other hand, by Lemma A.2.5, Δ_1, [$\bar{U}$/$\bar{X}$]$\Delta_2 \vdash$ [$\bar{U}$/$\bar{X}$]$\bar{T}$ <: [$\bar{U}$/$\bar{X}$][$\bar{T}$/$\bar{Y}$]$\bar{P}$. Since $\bar{Y}$ <: $\bar{P} \vdash \bar{P}$ by the rule GT-CLASS, $\bar{P}$ does not include any of $\bar{X}$ as a free variable. Thus, [$\bar{U}$/$\bar{X}$][$\bar{T}$/$\bar{Y}$]$\bar{P}$ = [[$\bar{U}$/$\bar{X}$]$\bar{T}$/$\bar{Y}$]$\bar{P}$, and finally, we have Δ_1, [$\bar{U}$/$\bar{X}$]$\Delta_2 \vdash$ C<[$\bar{U}$/$\bar{X}$]$\bar{T}$> ok by WF-CLASS. □

LEMMA A.2.7. *Suppose* $\Delta_1, \bar{\mathtt{X}} \mathrel{<:} \bar{\mathtt{N}}, \Delta_2 \vdash \mathtt{T}$ ok *and* $\Delta_1 \vdash \bar{\mathtt{U}} \mathrel{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$ *with* $\Delta_1 \vdash \bar{\mathtt{U}}$ ok *and none of* $\bar{\mathtt{X}}$ *appearing in* Δ_1. *Then,* $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash$ $\mathit{bound}_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}([\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}) \mathrel{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}](\mathit{bound}_{\Delta_1,\bar{\mathtt{X}} \mathrel{<:} \bar{\mathtt{N}},\Delta_2}(\mathtt{T}))$.

PROOF. The case where T is a nonvariable type is trivial. The case where T is a type variable X and $\mathtt{X} \in \mathit{dom}(\Delta_1) \cup \mathit{dom}(\Delta_2)$ is also easy. Finally, if T is a type variable $\mathtt{X}_i$, then $\mathit{bound}_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}([\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}) = \mathtt{U}_i$ and $[\bar{\mathtt{U}}/\bar{\mathtt{X}}](\mathit{bound}_{\Delta_1,\bar{\mathtt{X}} \mathrel{<:} \bar{\mathtt{N}},\Delta_2}(\mathtt{T})) =$ $[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{N}_i$; the assumption $\Delta_1 \vdash \bar{\mathtt{U}} \mathrel{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$ and Lemma A.2.1 finish the proof. □

LEMMA A.2.8. *If* $\Delta \vdash \mathtt{S} \mathrel{<:} \mathtt{T}$ *and* $\mathit{fields}(\mathit{bound}_\Delta(\mathtt{T})) = \bar{\mathtt{T}}\ \bar{\mathtt{f}}$, *then* $\mathit{fields}(\mathit{bound}_\Delta(\mathtt{S})) = \bar{\mathtt{S}}\ \bar{\mathtt{g}}$ *and* $\mathtt{S}_i = \mathtt{T}_i$ *and* $\mathtt{g}_i = \mathtt{f}_i$ *for all* $i \leq \#(\bar{\mathtt{f}})$.

PROOF. By straightforward induction on the derivation of $\Delta \vdash \mathtt{S} \mathrel{<:} \mathtt{T}$.

Case S-REFL. Trivial.
Case S-VAR. Trivial because $\mathit{bound}_\Delta(\mathtt{S}) = \mathit{bound}_\Delta(\mathtt{T})$.
Case S-TRANS. Easy.
Case S-CLASS. $\mathtt{S} = \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$ $\quad \mathtt{T} = [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{N}$
$\quad$ class $\mathtt{C}\texttt{<}\bar{\mathtt{X}} \lhd \bar{\mathtt{N}}\texttt{>} \lhd \mathtt{N}$ {$\bar{\mathtt{S}}\ \bar{\mathtt{g}}$; ...}

By the rule F-CLASS, $\mathit{fields}(\mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}) = \bar{\mathtt{U}}\ \bar{\mathtt{f}}, [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{S}}\ \bar{\mathtt{g}}$ *where* $\bar{\mathtt{U}}\ \bar{\mathtt{f}} = \mathit{fields}([\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{N})$. □

LEMMA A.2.9 *If* $\Delta \vdash \mathtt{T}$ ok *and* $\mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{T})) = \texttt{<}\bar{\mathtt{Y}} \lhd \bar{\mathtt{P}}\texttt{>}\bar{\mathtt{U}} \to \mathtt{U}_0$, *then for any* S *such that* $\Delta \vdash \mathtt{S} \mathrel{<:} \mathtt{T}$ *and* $\Delta \vdash \mathtt{S}$ ok, *we have* $\mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{S})) =$ $\texttt{<}\bar{\mathtt{Y}} \lhd \bar{\mathtt{P}}\texttt{>}\bar{\mathtt{U}} \to \mathtt{U}_0'$ *and* $\Delta, \bar{\mathtt{Y}} \mathrel{<:} \bar{\mathtt{P}} \vdash \mathtt{U}_0' \mathrel{<:} \mathtt{U}_0$.

PROOF. By straightforward induction on the derivation of $\Delta \vdash \mathtt{S} \mathrel{<:} \mathtt{T}$ with a case analysis by the last rule used.

Case S-REFL. Trivial.
Case S-VAR. Trivial because $\mathit{bound}_\Delta(\mathtt{S}) = \mathit{bound}_\Delta(\mathtt{T})$.
Case S-TRANS. Easy.
Case S-CLASS. $\mathtt{S} = \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$ $\quad \mathtt{T} = [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{N}$
$\quad$ class $\mathtt{C}\texttt{<}\bar{\mathtt{X}} \lhd \bar{\mathtt{N}}\texttt{>} \lhd \mathtt{N}$ {... $\bar{\mathtt{M}}$}

If $\bar{\mathtt{M}}$ do not include a declaration of m, it is easy to show the conclusion, since

$$\mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{S})) = \mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{T}))$$

by the rule MT-SUPER.

On the other hand, suppose $\bar{\mathtt{M}}$ includes a declaration of m. By straightforward induction on the derivation of $\mathit{mtype}(\mathtt{m}, \mathtt{T})$, we can show

$$\mathit{mtype}(\mathtt{m}, \mathtt{T}) = [\bar{\mathtt{T}}/\bar{\mathtt{X}}](\texttt{<}\bar{\mathtt{Y}} \lhd \bar{\mathtt{P}}'\texttt{>}\bar{\mathtt{U}}' \to \mathtt{U}_0'')$$

where $\texttt{<}\bar{\mathtt{Y}} \lhd \bar{\mathtt{P}}'\texttt{>}\bar{\mathtt{U}}' \to \mathtt{U}_0'' = \mathit{mtype}(\mathtt{m}, \mathtt{N})$. Without loss of generality, we can assume that $\bar{\mathtt{X}}$ and $\bar{\mathtt{Y}}$ are distinct and, in particular, that $[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{U}_0'' = \mathtt{U}_0$. By GT-METHOD, it must be the case that

$$\texttt{<}\bar{\mathtt{Y}} \lhd \bar{\mathtt{P}}'\texttt{>}\ \mathtt{W}_0'\ \mathtt{m}(\bar{\mathtt{U}}'\ \bar{\mathtt{x}})\{\ldots\} \in \bar{\mathtt{M}}$$

and

$$\bar{\mathtt{X}} \mathrel{<:} \bar{\mathtt{N}}, \bar{\mathtt{Y}} \mathrel{<:} \bar{\mathtt{P}}' \vdash \mathtt{W}_0' \mathrel{<:} \mathtt{U}_0''.$$

By Lemmas A.2.5 and A.2.1, we have

$$\Delta, \bar{\mathtt{Y}}\texttt{<:}\bar{\mathtt{P}} \vdash [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{W}_0{}'\texttt{<:}\mathtt{U}_0.$$

Since $\mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{S})) = \mathit{mtype}(\mathtt{m}, \mathtt{S}) = [\bar{\mathtt{T}}/\bar{\mathtt{X}}](\texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}'\texttt{>}\bar{\mathtt{U}}'\to\mathtt{W}_0{}')$ by MT-CLASS, letting $\mathtt{U}_0{}' = [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{W}_0{}'$ finishes the case. □

LEMMA A.2.10 (Type Substitution Preserves Typing). *If* $\Delta_1, \bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}}, \Delta_2;\ \Gamma \vdash \mathtt{e}:\mathtt{T}$ *and* $\Delta_1 \vdash \bar{\mathtt{U}} \texttt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$ *where* $\Delta_1 \vdash \bar{\mathtt{U}}$ ok *and none of* $\bar{\mathtt{X}}$ *appears in* Δ_1*, then* $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2; [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Gamma \vdash [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}:\mathtt{S}$ *for some* $\mathtt{S}$ *such that* $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \mathtt{S} \texttt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}$.

PROOF. By induction on the derivation of $\Delta_1, \bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}}, \Delta_2; \Gamma \vdash \mathtt{e}:\mathtt{T}$ with a case analysis on the last rule used.

Case GT-VAR. Trivial.
Case GT-FIELD. $\mathtt{e} = \mathtt{e}_0.\mathtt{f}_i \quad \Delta_1, \bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}}, \Delta_2; \Gamma \vdash \mathtt{e}_0:\mathtt{T}_0$
$\mathit{fields}(\mathit{bound}_{\Delta_1,\bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}},\Delta_2}(\mathtt{T}_0)) = \bar{\mathtt{T}}\ \bar{\mathtt{f}} \qquad \mathtt{T} = \mathtt{T}_i$

By the induction hypothesis, $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2; [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Gamma \vdash [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}_0:\mathtt{S}_0$ and $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \mathtt{S}_0 \texttt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_0$ for some $\mathtt{S}_0$. By Lemma A.2.7,

$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \mathit{bound}_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}([\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_0) \texttt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}](\mathit{bound}_{\Delta_1,\bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}},\Delta_2}(\mathtt{T}_0)).$$

Then, it is easy to show

$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \mathit{bound}_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}(\mathtt{S}_0) \texttt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}](\mathit{bound}_{\Delta_1,\bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}},\Delta_2}(\mathtt{T}_0)).$$

By Lemma A.2.8, $\mathit{fields}(\mathit{bound}_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}(\mathtt{S}_0)) = \bar{\mathtt{S}}\ \bar{\mathtt{g}}$, and we have $\mathtt{f}_j = \mathtt{g}_j$ and $\mathtt{S}_j = [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_j$ for $j \leq \#(\bar{\mathtt{f}})$. By the rule GT-FIELD, $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2; [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Gamma \vdash [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}_0.\mathtt{f}_i:\mathtt{S}_i$. Letting $\mathtt{S} = \mathtt{S}_i$ $(= [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_i)$ finishes the case.

Case GT-INVK. $\mathtt{e} = \mathtt{e}_0.\mathtt{m}\texttt{<}\bar{\mathtt{V}}\texttt{>}(\bar{\mathtt{e}}) \qquad \Delta_1, \bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}}, \Delta_2; \Gamma \vdash \mathtt{e}_0:\mathtt{T}_0$
$\mathit{mtype}(\mathtt{m}, \mathit{bound}_{\Delta_1,\bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}},\Delta_2}(\mathtt{T}_0)) = \texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\texttt{>}\bar{\mathtt{W}}\to\mathtt{W}_0$
$\Delta_1, \bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}}, \Delta_2 \vdash \bar{\mathtt{V}}$ ok $\quad \Delta_1, \bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}}, \Delta_2 \vdash \bar{\mathtt{V}} \texttt{<:} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}}$
$\Delta_1, \bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}}, \Delta_2; \Gamma \vdash \bar{\mathtt{e}}:\bar{\mathtt{S}}\ \Delta_1, \bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}}, \Delta_2 \vdash \bar{\mathtt{S}} \texttt{<:} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{W}}$
$\mathtt{T} = [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{W}_0$

By the induction hypothesis,

$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2; [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Gamma \vdash [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}_0:\mathtt{S}_0$$
$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \mathtt{S}_0 \texttt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_0$$

and

$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2; [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Gamma \vdash [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{e}}:\bar{\mathtt{S}}'$$
$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \bar{\mathtt{S}}' \texttt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{S}}.$$

By using Lemma A.2.7, it is easy to show

$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \mathit{bound}_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}(\mathtt{S}_0) \texttt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}](\mathit{bound}_{\Delta_1,\bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}},\Delta_2}(\mathtt{T}_0)).$$

Then, by Lemma A.2.9,

$$\mathit{mtype}(\mathtt{m}, \mathit{bound}_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}(\mathtt{S}_0)) = \texttt{<}\bar{\mathtt{Y}} \triangleleft [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{P}}\texttt{>}[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{W}}\to\mathtt{W}_0{}'$$
$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2, \bar{\mathtt{Y}}\texttt{<:}[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{P}} \vdash \mathtt{W}_0{}' \texttt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{W}_0.$$

By Lemma A.2.6,

$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{V}} \text{ ok}$$

Without loss of generality, we can assume that $\bar{X}$ and $\bar{Y}$ are distinct and that none of $\bar{Y}$ appear in $\bar{U}$; then $[\bar{U}/\bar{X}][\bar{V}/\bar{Y}] = [[\bar{U}/\bar{X}]\bar{V}/\bar{Y}][\bar{U}/\bar{X}]$. By Lemma A.2.5,

$$\begin{array}{ll}\Delta_1, [\bar{U}/\bar{X}]\Delta_2 \vdash [\bar{U}/\bar{X}]\bar{V} \mathrel{<:} [\bar{U}/\bar{X}][\bar{V}/\bar{Y}]\bar{P} & (= [[\bar{U}/\bar{X}]\bar{V}/\bar{Y}][\bar{U}/\bar{X}]\bar{P})\\ \Delta_1, [\bar{U}/\bar{X}]\Delta_2 \vdash [\bar{U}/\bar{X}]\bar{S} \mathrel{<:} [\bar{U}/\bar{X}][\bar{V}/\bar{Y}]\bar{W} & (= [[\bar{U}/\bar{X}]\bar{V}/\bar{Y}][\bar{U}/\bar{X}]\bar{W}).\end{array}$$

By the rule S-Trans,

$$\Delta_1, [\bar{U}/\bar{X}]\Delta_2 \vdash \bar{S}' \mathrel{<:} [[\bar{U}/\bar{X}]\bar{V}/\bar{Y}][\bar{U}/\bar{X}]\bar{W}.$$

By Lemma A.2.5, we have

$$\Delta_1, [\bar{U}/\bar{X}]\Delta_2 \vdash [\bar{V}/\bar{Y}]W_0{}' \mathrel{<:} [\bar{U}/\bar{X}][\bar{V}/\bar{Y}]W_0 \quad (= [[\bar{U}/\bar{X}]\bar{V}/\bar{Y}][\bar{U}/\bar{X}]W_0).$$

Finally, by the rule GT-Invk,

$$\Delta_1, [\bar{U}/\bar{X}]\Delta_2, [\bar{U}/\bar{X}]\Gamma \vdash ([\bar{U}/\bar{X}]e_0).m\texttt{<}[\bar{U}/\bar{X}]\bar{V}\texttt{>}([\bar{U}/\bar{X}]\bar{d}) : S$$

where $S = [\bar{V}/\bar{Y}]W_0{}'$, finishing the case.

Case GT-New, GT-UCast. Easy.

Case GT-DCast.

$$\begin{array}{lll} e = (N)\,e_0 & \Delta = \Delta_1, \bar{X}\mathrel{<:}\bar{N}, \Delta_2 & \\ \Delta;\Gamma \vdash e_0 : T_0 & \Delta \vdash N \mathrel{<:} \mathit{bound}_\Delta(T_0) & \\ N = C\texttt{<}\bar{T}\texttt{>} & \mathit{bound}_\Delta(T_0) = E\texttt{<}\bar{V}\texttt{>} & \mathit{dcast}(C, E)\end{array}$$

By the induction hypothesis, $\Delta_1, [\bar{U}/\bar{X}]\Delta_2; [\bar{U}/\bar{X}]\Gamma \vdash [\bar{U}/\bar{X}]e_0 : S_0$ for some S_0 such that $\Delta_1, [\bar{U}/\bar{X}]\Delta_2 \vdash S_0 \mathrel{<:} [\bar{U}/\bar{X}]T_0$. Let $\Delta' = \Delta_1, [\bar{U}/\bar{X}]\Delta_2$. We have three subcases according to a relation between S_0 and $[\bar{U}/\bar{X}]N$.

Subcase. $\Delta' \vdash \mathit{bound}_{\Delta'}(S_0) \mathrel{<:} [\bar{U}/\bar{X}]N$

By the rule GT-UCast, $\Delta';\Gamma \vdash [\bar{U}/\bar{X}]((N)\,e_0) : [\bar{U}/\bar{X}]N$.

Subcase. $\Delta' \vdash [\bar{U}/\bar{X}]N \mathrel{<:} \mathit{bound}_{\Delta'}(S_0) \quad [\bar{U}/\bar{X}]N \neq \mathit{bound}_{\Delta'}(S_0)$

By using Lemma A.2.7 and the fact that $\Delta \vdash S \mathrel{<:} T$ implies $\Delta \vdash \mathit{bound}_\Delta(S) \mathrel{<:} \mathit{bound}_\Delta(T)$, we have $\Delta' \vdash \mathit{bound}_{\Delta'}(S_0) \mathrel{<:} [\bar{U}/\bar{X}]\mathit{bound}_\Delta(T_0)$. Then, $C \trianglelefteq D \trianglelefteq E$ where $\mathit{bound}_{\Delta'}(S_0) = D\texttt{<}\bar{W}\texttt{>}$. If $C \neq D \neq E$, we have, by Lemma A.2.4, $\mathit{dcast}(C, D)$; the rule GT-DCast finishes the subcase. The case $C = D$ cannot happen, since it implies $[\bar{U}/\bar{X}]N = \mathit{bound}_{\Delta'}(S_0)$. Finally, the other case $D = E$ is trivial.

Subcase. $\Delta' \vdash [\bar{U}/\bar{X}]N \not\mathrel{<:} \mathit{bound}_{\Delta'}(S_0) \quad \Delta' \vdash \mathit{bound}_{\Delta'}(S_0) \not\mathrel{<:} [\bar{U}/\bar{X}]N$

By using Lemma A.2.7 and the fact that $\Delta' \vdash S \mathrel{<:} T$ implies $\Delta' \vdash \mathit{bound}_{\Delta'}(S) \mathrel{<:} \mathit{bound}_{\Delta'}(T)$, we have $\Delta' \vdash \mathit{bound}_{\Delta'}(S_0) \mathrel{<:} [\bar{U}/\bar{X}](\mathit{bound}_\Delta(T_0))$.

Let $\mathit{bound}_{\Delta'}(S_0) = D\texttt{<}\bar{W}\texttt{>}$. We show below that, by contradiction, neither $C \trianglelefteq D$ nor $D \trianglelefteq C$ holds. Suppose $C \trianglelefteq D$. Then, there exist some $\bar{V}'$ such that $\Delta' \vdash C\texttt{<}\bar{V}'\texttt{>} \mathrel{<:} \mathit{bound}_{\Delta'}(S_0)$. By Lemma A.2.4, we have $\mathit{dcast}(C, D)$; it follows from Lemma A.2.3 that $C\texttt{<}V'\texttt{>} = [\bar{U}/\bar{X}]N$, contradicting the assumption $\Delta' \vdash [\bar{U}/\bar{X}]N \not\mathrel{<:} \mathit{bound}_{\Delta'}(S_0)$; thus, $C \not\trianglelefteq D$. On the other hand, suppose $D \trianglelefteq C$. Since we have $\Delta' \vdash \mathit{bound}_{\Delta'}(S_0) \mathrel{<:} [\bar{U}/\bar{X}](\mathit{bound}_\Delta(T_0))$, we can have $C\texttt{<}\bar{V}'\texttt{>}$ such that $\Delta' \vdash \mathit{bound}_{\Delta'}(S_0) \mathrel{<:} C\texttt{<}\bar{V}'\texttt{>}$ and $\Delta' \vdash C\texttt{<}\bar{V}'\texttt{>} \mathrel{<:} [\bar{U}/\bar{X}](\mathit{bound}_\Delta(T_0))$. Then, $[\bar{U}/\bar{X}]N = C\texttt{<}\bar{V}'\texttt{>}$ by Lemma A.2.3, contradicting the assumption $\Delta' \vdash \mathit{bound}_{\Delta'}(S_0) \not\mathrel{<:} [\bar{U}/\bar{X}]N$; thus, $D \not\trianglelefteq C$.

Finally, by the rule GT-SCAST, $\Delta;\Gamma \vdash [\bar{\mathtt{T}}/\bar{\mathtt{X}}]((\mathtt{N})\,\mathtt{e}_0) : [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{N}$ with *stupid warning*.

Case GT-SCAST. $\mathtt{e} = (\mathtt{N})\,\mathtt{e}_0 \quad \Delta = \Delta_1, \bar{\mathtt{X}}\mathtt{<:}\bar{\mathtt{N}}, \Delta_2 \quad \Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0$
$\mathtt{N} = \mathtt{C<}\bar{\mathtt{T}}\mathtt{>} \quad \mathit{bound}_\Delta(\mathtt{T}_0) = \mathtt{E<}\bar{\mathtt{V}}\mathtt{>} \quad \mathtt{C} \not\trianglelefteq \mathtt{E} \quad \mathtt{E} \not\trianglelefteq \mathtt{C}$

By the induction hypothesis, $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2; [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Gamma \vdash [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}_0 : \mathtt{S}_0$ for some $\mathtt{S}_0$ such that $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \mathtt{S}_0 \mathtt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_0$. Using Lemma A.2.7, we have $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \mathit{bound}_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}(\mathtt{S}_0) \mathtt{<:} [\bar{\mathtt{U}}/\bar{\mathtt{X}}](\mathit{bound}_\Delta(\mathtt{T}_0))$. Let $\mathit{bound}_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}(\mathtt{S}_0) = \mathtt{D<}\bar{\mathtt{W}}\mathtt{>}$. Since it is the case that $[\bar{\mathtt{U}}/\bar{\mathtt{X}}](\mathit{bound}_\Delta(\mathtt{T}_0)) = \mathtt{E<}[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{V}}\mathtt{>}$, by Lemma A.2.2, $\mathtt{D} \not\trianglelefteq \mathtt{C}$ and $\mathtt{C} \not\trianglelefteq \mathtt{D}$. By the rule GT-SCAST, $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2; [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Gamma \vdash [\bar{\mathtt{U}}/\bar{\mathtt{X}}](\mathtt{N})\,\mathtt{e}_0 : [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{N}$ with *stupid warning*, finishing the case. □

LEMMA A.2.11 (Term Substitution Preserves Typing). *If* $\Delta;\Gamma, \bar{\mathtt{x}} : \bar{\mathtt{T}} \vdash \mathtt{e} : \mathtt{T}$ *and* $\Delta;\Gamma \vdash \bar{\mathtt{d}} : \bar{\mathtt{S}}$ *where* $\Delta \vdash \bar{\mathtt{S}}\mathtt{<:}\bar{\mathtt{T}}$*, then* $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e} : \mathtt{S}$ *for some* $\mathtt{S}$ *such that* $\Delta \vdash \mathtt{S}\mathtt{<:}\mathtt{T}$.

PROOF. By induction on the derivation of $\Delta;\Gamma, \bar{\mathtt{x}} : \bar{\mathtt{T}} \vdash \mathtt{e} : \mathtt{T}$ with a case analysis on the last rule used.

Case GT-VAR. $\mathtt{e} = \mathtt{x}$

If $\mathtt{x} \in \mathit{dom}(\Gamma)$, then the conclusion is immediate, since $[\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{x} = \mathtt{x}$. On the other hand, if $\mathtt{x} = \mathtt{x}_i$ and $\mathtt{T} = \mathtt{T}_i$, then letting $\mathtt{S} = \mathtt{S}_i$ finishes the case.

Case GT-FIELD. $\mathtt{e} = \mathtt{e}_0.\mathtt{f}_i \quad \Delta;\Gamma, \bar{\mathtt{x}} : \bar{\mathtt{T}} \vdash \mathtt{e}_0 : \mathtt{T}_0$
$\mathit{fields}(\mathit{bound}_\Delta(\mathtt{T}_0)) = \bar{\mathtt{T}}\ \bar{\mathtt{f}} \quad \mathtt{T} = \mathtt{T}_i$

By the induction hypothesis, $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0 : \mathtt{S}_0$ for some $\mathtt{S}_0$ such that $\Delta \vdash \mathtt{S}_0 \mathtt{<:} \mathtt{T}_0$. By Lemma A.2.8, $\mathit{fields}(\mathit{bound}_\Delta(\mathtt{S}_0)) = \bar{\mathtt{S}}\ \bar{\mathtt{g}}$ such that $\mathtt{S}_j = \mathtt{T}_j$ and $\mathtt{f}_j = \mathtt{g}_j$ for all $j \leq \#(\bar{\mathtt{T}})$. Therefore, by the rule GT-FIELD, $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0.\mathtt{f}_i : \mathtt{T}$

Case GT-INVK. $\mathtt{e} = \mathtt{e}_0.\mathtt{m<}\bar{\mathtt{V}}\mathtt{>}(\bar{\mathtt{e}}) \quad \Delta;\Gamma, \bar{\mathtt{x}} : \bar{\mathtt{T}} \vdash \mathtt{e}_0 : \mathtt{T}_0$
$\mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{T}_0)) = \mathtt{<}\bar{\mathtt{Y}} \trianglelefteq \bar{\mathtt{P}}\mathtt{>}\bar{\mathtt{U}}\rightarrow\mathtt{U} \quad \Delta \vdash \bar{\mathtt{V}}\ \text{ok}$
$\Delta \vdash \bar{\mathtt{V}} \mathtt{<:} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}} \quad \Delta;\Gamma, \bar{\mathtt{x}} : \bar{\mathtt{T}} \vdash \bar{\mathtt{e}} : \bar{\mathtt{S}}$
$\Delta \vdash \bar{\mathtt{S}} \mathtt{<:} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}} \quad \mathtt{T} = [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}$

By the induction hypothesis, $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0 : \mathtt{S}_0$ for some $\mathtt{S}_0$ such that $\Delta \vdash \mathtt{S}_0 \mathtt{<:} \mathtt{T}_0$ and $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\bar{\mathtt{e}} : \bar{\mathtt{W}}$ for some $\bar{\mathtt{W}}$ such that $\Delta \vdash \bar{\mathtt{W}} \mathtt{<:} \bar{\mathtt{S}}$. By Lemma A.2.9, $\mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{S}_0)) = \mathtt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\mathtt{>}\bar{\mathtt{U}}\rightarrow\mathtt{U}'$ and $\Delta, \bar{\mathtt{Y}}\mathtt{<:}\bar{\mathtt{P}} \vdash \mathtt{U}'\mathtt{<:}\mathtt{U}$. By Lemma A.2.5, $\Delta \vdash [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}'\mathtt{<:}[\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}$. By the rule GT-METHOD, $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}](\mathtt{e}_0.\mathtt{m<}\bar{\mathtt{V}}\mathtt{>}(\bar{\mathtt{e}})) : [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}'$. Letting $\mathtt{S} = [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}'$ finishes the case.

*Case*GT-NEW, GT-UCAST. Easy.
Case GT-DCAST. $\mathtt{e} = (\mathtt{N})\,\mathtt{e}_0 \quad \Delta;\Gamma, \bar{\mathtt{x}} : \bar{\mathtt{T}} \vdash \mathtt{e}_0 : \mathtt{T}_0$
$\Delta \vdash \mathtt{N} \mathtt{<:} \mathit{bound}_\Delta(\mathtt{T}_0) \quad \mathtt{N} = \mathtt{C<}\bar{\mathtt{U}}\mathtt{>}$
$\mathit{bound}_\Delta(\mathtt{T}_0) = \mathtt{E<}\bar{\mathtt{V}}\mathtt{>} \quad \mathit{dcast}(\mathtt{C}, \mathtt{E})$

By the induction hypothesis, $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0 : \mathtt{S}_0$ for some $\mathtt{S}_0$ such that $\Delta \vdash \mathtt{S}_0 \mathtt{<:} \mathtt{T}_0$. We have three subcases according to a relation between $\mathtt{S}_0$ and $\mathtt{N}$.

Subcase. $\Delta \vdash \mathit{bound}_\Delta(\mathtt{S}_0) \mathtt{<:} \mathtt{N}$

By the rule GT-UCAST, $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]((\mathtt{N})\,\mathtt{e}_0):\mathtt{N}$.

Subcase. $\Delta \vdash \mathtt{N} <: \mathit{bound}_\Delta(\mathtt{S}_0) \quad \mathtt{N} \neq \mathit{bound}_\Delta(\mathtt{S}_0)$

By using Lemma A.2.7 and the fact that $\Delta \vdash \mathtt{S} <: \mathtt{T}$ implies $\Delta \vdash \mathit{bound}_\Delta(\mathtt{S}) <: \mathit{bound}_\Delta(\mathtt{T})$, we have $\Delta \vdash \mathit{bound}_\Delta(\mathtt{S}_0) <: \mathit{bound}_\Delta(\mathtt{T}_0)$. Then, $\mathtt{C} \trianglelefteq \mathtt{D} \trianglelefteq \mathtt{E}$ where $\mathit{bound}_\Delta(\mathtt{S}_0) = \mathtt{D}\texttt{<}\bar{\mathtt{W}}\texttt{>}$. If $\mathtt{C} \neq \mathtt{D} \neq \mathtt{E}$, we have, by Lemma A.2.4, $\mathit{dcast}(\mathtt{C},\mathtt{D})$; the rule GT-DCAST finishes the subcase. The case $\mathtt{C} = \mathtt{D}$ cannot happen, since it implies $\mathtt{N} = \mathit{bound}_\Delta(\mathtt{S}_0)$, and the other case, $\mathtt{D} = \mathtt{E}$, is trivial.

Subcase. $\Delta \vdash \mathtt{N} \not<: \mathit{bound}_\Delta(\mathtt{S}_0) \quad \Delta \vdash \mathit{bound}_\Delta(\mathtt{S}_0) \not<: \mathtt{N}$

Let $\mathit{bound}_\Delta(\mathtt{S}_0) = \mathtt{D}\texttt{<}\bar{\mathtt{W}}\texttt{>}$. We show that, by contradiction, $\mathtt{C} \not\trianglelefteq \mathtt{D}$ and $\mathtt{D} \not\trianglelefteq \mathtt{C}$.

Suppose $\mathtt{C} \trianglelefteq \mathtt{D}$. Then, we can have $\mathtt{C}\texttt{<}\bar{\mathtt{U}}'\texttt{>}$ such that $\Delta \vdash \mathtt{C}\texttt{<}\bar{\mathtt{U}}'\texttt{>} <: \mathtt{D}\texttt{<}\bar{\mathtt{W}}\texttt{>}$. By transitivity of $<:$ and the fact that $\Delta \vdash \mathtt{S}_0 <: \mathtt{T}_0$ implies $\Delta \vdash \mathit{bound}_\Delta(\mathtt{S}_0) <: \mathit{bound}_\Delta(\mathtt{T}_0)$, we have $\Delta \vdash \mathtt{C}\texttt{<}\bar{\mathtt{U}}'\texttt{>} <: \mathit{bound}_\Delta(\mathtt{T}_0)$. Thus, $\bar{\mathtt{U}}' = \bar{\mathtt{U}}$, contradicting the assumption $\Delta \vdash \mathtt{N} \not<: \mathit{bound}_\Delta(\mathtt{S}_0)$ $(= \mathtt{D}\texttt{<}\bar{\mathtt{W}}\texttt{>})$. On the other hand, suppose $\mathtt{D} \trianglelefteq \mathtt{C}$. Since we have $\Delta \vdash \mathit{bound}_\Delta(\mathtt{S}_0) <: \mathit{bound}_\Delta(\mathtt{T}_0)$, we can have $\mathtt{C}\texttt{<}\bar{\mathtt{V}}'\texttt{>}$ such that $\Delta \vdash \mathit{bound}_\Delta(\mathtt{S}_0) <: \mathtt{C}\texttt{<}\bar{\mathtt{V}}'\texttt{>}$ and $\Delta' \vdash \mathtt{C}\texttt{<}\bar{\mathtt{V}}'\texttt{>} <: \mathit{bound}_\Delta(\mathtt{T}_0)$. Then, $\mathtt{N} = \mathtt{C}\texttt{<}\bar{\mathtt{V}}'\texttt{>}$ by Lemma A.2.3, contradicting the assumption $\Delta \vdash \mathit{bound}_\Delta(\mathtt{S}_0) <: \mathtt{N}$; thus, $\mathtt{D} \not\trianglelefteq \mathtt{C}$.

Finally, by the rule GT-SCAST, $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]((\mathtt{N})\,\mathtt{e}_0):\mathtt{N}$ with *stupid warning*.

Case GT-SCAST. $\quad \Delta;\Gamma,\bar{\mathtt{x}}:\bar{\mathtt{T}} \vdash \mathtt{e}_0:\mathtt{T}_0 \quad \mathtt{N} = \mathtt{C}\texttt{<}\bar{\mathtt{U}}\texttt{>} \quad \mathit{bound}_\Delta(\mathtt{T}_0) = \mathtt{E}\texttt{<}\bar{\mathtt{V}}\texttt{>}$
$\quad \mathtt{C} \not\trianglelefteq \mathtt{E} \quad \mathtt{E} \not\trianglelefteq \mathtt{C}$

By the induction hypothesis, $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}_0:\mathtt{S}_0$ for some $\mathtt{S}_0$ such that $\Delta \vdash \mathtt{S}_0 <: \mathtt{T}_0$, which implies $\Delta \vdash \mathit{bound}_\Delta(\mathtt{S}_0) <: \mathit{bound}_\Delta(\mathtt{T}_0)$. Let $\mathit{bound}_\Delta(\mathtt{S}_0) = \mathtt{D}\texttt{<}\bar{\mathtt{W}}\texttt{>}$. By Lemma A.2.2, we have $\mathtt{D} \not\trianglelefteq \mathtt{C}$ and $\mathtt{C} \not\trianglelefteq \mathtt{D}$. Then, by the rule GT-SCAST, $\Delta;\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]((\mathtt{N})\,\mathtt{e}_0):\mathtt{N}$ again with *stupid warning*. □

LEMMA A.2.12 *If* $\mathit{mtype}(\mathtt{m},\mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}) = \texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\texttt{>}\bar{\mathtt{U}}\rightarrow\mathtt{U}$ *and* $\mathit{mbody}(\mathtt{m}\texttt{<}\bar{\mathtt{V}}\texttt{>},\mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}) = \bar{\mathtt{x}}.\mathtt{e}_0$ *where* $\Delta \vdash \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$ ok *and* $\Delta \vdash \bar{\mathtt{V}}$ ok *and* $\Delta \vdash \bar{\mathtt{V}} <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}}$, *then there exist some* $\mathtt{N}$ *and* $\mathtt{S}$ *such that* $\Delta \vdash \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>} <: \mathtt{N}$ *and* $\Delta \vdash \mathtt{N}$ ok *and* $\Delta \vdash \mathtt{S} <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}$ *and* $\Delta \vdash \mathtt{S}$ ok *and* $\Delta;\bar{\mathtt{x}}:[\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}},\mathtt{this}:\mathtt{N} \vdash \mathtt{e}_0:\mathtt{S}$.

PROOF. By induction on the derivation of $\mathit{mbody}(\mathtt{m}\texttt{<}\bar{\mathtt{V}}\texttt{>},\mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}) = \bar{\mathtt{x}}.\mathtt{e}$ using Lemmas A.2.5 and A.2.10.

Case MB-CLASS. $\quad \mathtt{class\ C}\texttt{<}\bar{\mathtt{X}} \triangleleft \bar{\mathtt{N}}\texttt{>} \triangleleft \mathtt{P}\ \{\ldots\ \bar{\mathtt{M}}\}$
$\quad \texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{Q}}\texttt{>}\ \mathtt{T}_0\ \mathtt{m}(\bar{\mathtt{S}}\ \bar{\mathtt{x}})\{\ \mathtt{return\ e;}\ \} \in \bar{\mathtt{M}}$

Let $\Gamma = \bar{\mathtt{x}}:\bar{\mathtt{S}},\mathtt{this}:\mathtt{C}\texttt{<}\bar{\mathtt{X}}\texttt{>}$ and $\Delta' = \bar{\mathtt{X}}<:\bar{\mathtt{N}},\bar{\mathtt{Y}}<:\bar{\mathtt{Q}}$. By the rules GT-CLASS and GT-METHOD, we have $\Delta';\Gamma \vdash \mathtt{e}:\mathtt{S}_0$ and $\Delta';\Gamma \vdash \mathtt{S}_0 <: \mathtt{T}_0$ for some $\mathtt{S}_0$. Since $\Delta \vdash \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$ ok, we have $\Delta \vdash \bar{\mathtt{T}} <: [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$ by the rule WF-CLASS. By Lemmas A.2.1, A.2.5, and A.2.10,

$$\Delta,\bar{\mathtt{Y}}<:[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{Q}} \vdash [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{S}_0 <: [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{T}_0$$

and

$$\Delta,\bar{\mathtt{Y}}<:[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{Q}};\bar{\mathtt{x}}:[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{S}},\mathtt{this}:\mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>} \vdash [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{e}:\mathtt{S}_0{}'$$

where

$$\Delta,\bar{\mathtt{Y}}<:[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{Q}} \vdash \mathtt{S}_0{}' <: [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{S}_0.$$

Now, we can assume $\bar{\mathtt{X}}$ and $\bar{\mathtt{Y}}$ are distinct without loss of generality. By the rule MT-Class, we have

$$[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{Q}} = \bar{\mathtt{P}} \qquad [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{S}} = \bar{\mathtt{U}} \qquad [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{T}_0 = \mathtt{U}.$$

Again, by rule S-Trans and Lemmas A.2.5 and A.2.10,

$$\Delta \vdash [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{S_0}' <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}$$

and

$$\Delta; \bar{\mathtt{x}} : [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}}, \mathtt{this} : \mathtt{C}\langle\bar{\mathtt{T}}\rangle \vdash [\bar{\mathtt{V}}/\bar{\mathtt{Y}}][\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{e} : \mathtt{S_0}''$$

where

$$\Delta \vdash \mathtt{S_0}'' <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{S_0}'.$$

Since we can assume that any of $\bar{\mathtt{Y}}$ does not occur in $\bar{\mathtt{T}}$ without loss of generality,

$$\mathtt{e}_0 = [\bar{\mathtt{T}}/\bar{\mathtt{X}}, \bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{e} = [\bar{\mathtt{V}}/\bar{\mathtt{Y}}][\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{e}.$$

Letting $\mathtt{N} = \mathtt{C}\langle\bar{\mathtt{T}}\rangle$ and $\mathtt{S} = \mathtt{S_0}''$ finishes the case.

Case MB-Super. $\mathtt{class}\ \mathtt{C}\langle\bar{\mathtt{X}} \triangleleft \bar{\mathtt{N}}\rangle \triangleleft \mathtt{N}\ \{\ldots\ \bar{\mathtt{M}}\} \quad \mathtt{m} \not\in \bar{\mathtt{M}}$

Immediate from the induction hypothesis and the fact that $\Delta \vdash \mathtt{C}\langle\bar{\mathtt{T}}\rangle <: [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{N}$. □

Proof of Theorem 3.4.1. By induction on the derivation of $\mathtt{e} \rightarrow \mathtt{e}'$ with a case analysis on the reduction rule used. We show the main cases.

Case GR-Field. $\mathtt{e} = \mathtt{new}\ \mathtt{N}(\bar{\mathtt{e}}).\mathtt{f}_i \quad \mathit{fields}(\mathtt{N}) = \bar{\mathtt{T}}\ \bar{\mathtt{f}} \quad \mathtt{e}' = \mathtt{e}_i$

By the rules GT-Field and GT-New, we have

$$\begin{array}{l} \Delta; \Gamma \vdash \mathtt{new}\ \mathtt{N}(\bar{\mathtt{e}}) : \mathtt{N} \\ \Delta; \Gamma \vdash \bar{\mathtt{e}} : \bar{\mathtt{S}} \\ \Delta \vdash \bar{\mathtt{S}} <: \bar{\mathtt{T}}. \end{array}$$

In particular, $\Delta; \Gamma \vdash \mathtt{e}_i : \mathtt{S}_i$ finishes the case.

Case GR-Invk. $\mathtt{e} = \mathtt{new}\ \mathtt{N}(\bar{\mathtt{e}}).\mathtt{m}\langle\bar{\mathtt{V}}\rangle(\bar{\mathtt{d}}) \qquad \mathit{mbody}(\mathtt{m}\langle\bar{\mathtt{V}}\rangle, \mathtt{N}) = \bar{\mathtt{x}}.\mathtt{e}_0$
$\mathtt{e}' = [\bar{\mathtt{d}}/\bar{\mathtt{x}}, \mathtt{new}\ \mathtt{N}(\bar{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0$

By the rules GT-Invk and GT-New, we have

$$\begin{array}{ll} \Delta; \Gamma \vdash \mathtt{new}\ \mathtt{N}(\bar{\mathtt{e}}) : \mathtt{N} & \mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{N})) = \langle\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\rangle\bar{\mathtt{U}}\rightarrow\mathtt{U} \\ \Delta \vdash \bar{\mathtt{V}}\ \mathrm{ok} & \Delta \vdash \bar{\mathtt{V}} <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}} \\ \Delta; \Gamma \vdash \bar{\mathtt{d}} : \bar{\mathtt{S}} & \Delta \vdash \bar{\mathtt{S}} <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}} \\ \mathtt{T} = [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U} & \Delta \vdash \mathtt{N}\ \mathrm{ok} \end{array}$$

By Lemma A.2.12, $\Delta; \bar{\mathtt{x}} : [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}}, \mathtt{this} : \mathtt{P} \vdash \mathtt{e}_0 : \mathtt{S}$ for some $\mathtt{P}$ and $\mathtt{S}$ such that $\Delta \vdash \mathtt{N} <: \mathtt{P}$ where $\Delta \vdash \mathtt{P}$ ok, and $\Delta \vdash \mathtt{S} <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}$ where $\Delta \vdash \mathtt{S}$ ok. Then, by Lemmas A.2.1 and A.2.11, $\Delta; \Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}, \mathtt{new}\ \mathtt{N}(\bar{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0 : \mathtt{T}_0$ for some $\mathtt{T}_0$ such that $\Delta \vdash \mathtt{T}_0 <: \mathtt{S}$. By the rule S-Trans, we have $\Delta \vdash \mathtt{T}_0 <: \mathtt{T}$. Finally, letting $\mathtt{T}' = \mathtt{T}_0$ finishes the case.

Case GR-Cast. Easy.

Case GRC-Field. $\mathtt{e} = \mathtt{e}_0.\mathtt{f} \quad \mathtt{e}' = \mathtt{e}_0'.\mathtt{f} \qquad \mathtt{e}_0 \rightarrow \mathtt{e}_0'$

By the rule GT-FIELD, we have

$$\begin{array}{l} \Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \\ \mathit{fields}(\mathit{bound}_\Delta(\mathtt{T}_0)) = \bar{\mathtt{T}}\ \bar{\mathtt{f}} \\ \mathtt{T} = \mathtt{T}_i \end{array}$$

By the induction hypothesis, $\Delta;\Gamma \vdash \mathtt{e}_0' : \mathtt{T}_0'$ for some $\mathtt{T}_0'$ such that $\Delta \vdash \mathtt{T}_0' \mathrel{<:} \mathtt{T}_0$. By Lemma A.2.8, $\mathit{fields}(\mathit{bound}_\Delta(\mathtt{T}_0')) = \bar{\mathtt{T}}'\ \bar{\mathtt{g}}$ and, for $j \le \#(\bar{\mathtt{f}})$, we have $\mathtt{g}_i = \mathtt{f}_i$ and $\mathtt{T}_i' = \mathtt{T}_i$. Therefore, by the rule GT-FIELD, $\Delta;\Gamma \vdash \mathtt{e}_0'.\mathtt{f} : \mathtt{T}_i'$. Letting $\mathtt{T}' = \mathtt{T}_i'$ finishes the case.

Case GRC-INV-RECV. $\mathtt{e} = \mathtt{e}_0.\mathtt{m}\texttt{<}\bar{\mathtt{V}}\texttt{>}(\bar{\mathtt{e}})$ $\quad \mathtt{e}' = \mathtt{e}_0'.\mathtt{m}\texttt{<}\bar{\mathtt{V}}\texttt{>}(\bar{\mathtt{e}})$
$\mathtt{e}_0 \to \mathtt{e}_0'$

By the rule GT-INVK, we have

$$\begin{array}{ll} \Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 & \mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{T}_0)) = \texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\texttt{>}\bar{\mathtt{T}} \to \mathtt{U} \\ \Delta \vdash \bar{\mathtt{V}}\ \text{ok} & \Delta \vdash \bar{\mathtt{V}} \mathrel{<:} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}} \\ \Delta \vdash \bar{\mathtt{e}} : \bar{\mathtt{S}} & \Delta \vdash \bar{\mathtt{S}} \mathrel{<:} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{T}} \\ \mathtt{T} = [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U} & \end{array}$$

By the induction hypothesis, $\Delta;\Gamma \vdash \mathtt{e}_0' : \mathtt{T}_0'$ for some $\mathtt{T}_0'$ such that $\Delta \vdash \mathtt{T}_0' \mathrel{<:} \mathtt{T}_0$. By Lemma A.2.9, $\mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{T}_0')) = \texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\texttt{>}\bar{\mathtt{T}} \to \mathtt{V}$ and $\Delta, \bar{\mathtt{Y}} \mathrel{<:} \bar{\mathtt{P}} \vdash \mathtt{V} \mathrel{<:} \mathtt{U}$. By Lemma A.2.5, $\Delta \vdash [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{V} \mathrel{<:} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}$. Then, by the rule GT-INVK, $\Delta;\Gamma \vdash \mathtt{e}_0'.\mathtt{m}\texttt{<}\bar{\mathtt{V}}\texttt{>}(\bar{\mathtt{e}}) : [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{V}$. Letting $\mathtt{T}_0' = [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{V}$ finishes the case.

Case GRC-CAST. $\mathtt{e} = (\mathtt{N})\mathtt{e}_0 \quad \mathtt{e}' = (\mathtt{N})\mathtt{e}_0' \quad \mathtt{e}_0 \to \mathtt{e}_0'$

There are three subcases according to the last typing rule: GT-UCAST, GT-DCAST, and GT-SCAST. These subcases are similar to the subcases in the case for GT-DCAST in the proof of Lemma A.2.11.

Case GRC-INV-ARG, GRC-NEW-ARG. Easy. □

A.3 Proof of Theorem 4.5.1

First, we show that if an expression is well typed, then its type is well formed (Lemma A.3.4). Note that we assume that the underlying GJ class table *CT* is ok.

LEMMA A.3.1. *If* $\Delta \vdash \mathtt{S} \mathrel{<:} \mathtt{T}$ *and* $\Delta \vdash \mathtt{S}$ ok *for some well-formed type environment* Δ, *then* $\Delta \vdash \mathtt{T}$ ok.

PROOF. By induction on the derivation of $\Delta \vdash \mathtt{S} \mathrel{<:} \mathtt{T}$ with a case analysis on the last rule used. The cases for S-REFL and S-TRANS are easy.

Case S-VAR. $\mathtt{S} = \mathtt{X} \qquad \mathtt{T} = \Delta(\mathtt{X})$

$\mathtt{T}$ must be well formed, since Δ is well formed.

Case S-CLASS. $\mathtt{S} = \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>} \qquad \mathtt{T} = [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{N}$
$\mathtt{class}\ \mathtt{C}\texttt{<}\bar{\mathtt{X}} \triangleleft \bar{\mathtt{N}}\texttt{>} \triangleleft \mathtt{N}\ \{\ldots\}$
$\Delta \vdash \bar{\mathtt{T}}$ ok $\qquad \Delta \vdash \bar{\mathtt{T}} \mathrel{<:} [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$

Since *CT*(C) is ok, we also have $\bar{\mathtt{X}} \mathrel{<:} \bar{\mathtt{N}} \vdash \mathtt{N}$ ok by the rule GT-CLASS. Then, by Lemmas A.2.1 and A.2.6, $\Delta \vdash [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{N}$ ok. □

LEMMA A.3.2. *If* $\mathit{fields}_{\mathrm{FGJ}}(\mathtt{N}) = \bar{\mathtt{U}}\ \bar{\mathtt{f}}$ *and* $\Delta \vdash \mathtt{N}$ ok *for some well-formed type environment* Δ*, then* $\Delta \vdash \bar{\mathtt{U}}$ ok.

PROOF. By induction on the derivation of $\mathit{fields}_{\mathrm{FGJ}}(\mathtt{N})$ with a case analysis on the last rule used.

Case F-OBJECT. Trivial.
Case F-CLASS. $\mathtt{N} = \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$
$\mathtt{class\ C}\texttt{<}\bar{\mathtt{X}} \triangleleft \bar{\mathtt{N}}\texttt{>} \triangleleft \mathtt{P}\ \{\bar{\mathtt{S}}\ \bar{\mathtt{g}}';\ \mathtt{K}\ \bar{\mathtt{M}}\}$
$\mathit{fields}_{\mathrm{FGJ}}([\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{P}) = \bar{\mathtt{V}}\ \bar{\mathtt{g}}$
$\bar{\mathtt{U}}\ \bar{\mathtt{f}} = \bar{\mathtt{V}}\ \bar{\mathtt{g}}, [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{S}}\ \bar{\mathtt{g}}'$

Since $\mathit{CT}(\mathtt{C})$ is ok, by the rule GT-CLASS, $\bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}} \vdash \mathtt{P}$ ok. By Lemmas A.2.1 and A.2.6, $\Delta \vdash [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{P}$ ok. Then, by the induction hypothesis, $\Delta \vdash \bar{\mathtt{V}}$ ok. Since $\Delta \vdash \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$ ok, we have $\Delta \vdash \bar{\mathtt{T}}$ ok and $\Delta \vdash \bar{\mathtt{T}} \texttt{<:} [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$ by the rule WF-CLASS. On the other hand, by the rule GT-CLASS, we have $\bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}} \vdash \bar{\mathtt{S}}$ ok. Finally, by Lemmas A.2.1 and A.2.6, $\Delta \vdash [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{S}}$ ok, finishing the case. □

LEMMA A.3.3. *If* $\mathit{mtype}_{\mathrm{FGJ}}(\mathtt{m}, \mathtt{N}) = \texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\texttt{>}\bar{\mathtt{U}}\rightarrow\mathtt{U}_0$ *and* $\Delta \vdash \mathtt{N}$ ok *for some well-formed type environment* Δ*, then* $\Delta, \bar{\mathtt{Y}}\texttt{<:}\bar{\mathtt{P}} \vdash \mathtt{U}_0$ ok.

PROOF. By induction on the derivation of $\mathit{mtype}_{\mathrm{FGJ}}(\mathtt{m}, \mathtt{N})$ with a case analysis on the last rule used.

Case MT-CLASS. $\mathtt{N} = \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$
$\mathtt{class\ C}\texttt{<}\bar{\mathtt{X}} \triangleleft \bar{\mathtt{N}}\texttt{>} \triangleleft \mathtt{P}\ \{\ldots\ \bar{\mathtt{M}}\}$
$\texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{Q}}\texttt{>}\ \mathtt{S}_0\ \mathtt{m}(\bar{\mathtt{S}}\ \bar{\mathtt{x}})\{\ \mathtt{return\ e}_0;\ \} \in \bar{\mathtt{M}}$
$[\bar{\mathtt{T}}/\bar{\mathtt{X}}](\texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{Q}}\texttt{>}\bar{\mathtt{S}}\rightarrow\mathtt{S}_0) = \texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\texttt{>}\bar{\mathtt{U}} \rightarrow \mathtt{U}_0$

Without loss of generality, we can assume that $\bar{\mathtt{X}}$ and $\bar{\mathtt{Y}}$ are distinct and that $[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{Q}} = \bar{\mathtt{P}}$ and $[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{S}_0 = \mathtt{U}_0$. By the rules GT-CLASS and GT-METHOD, we have

$$\bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}}, \bar{\mathtt{Y}}\texttt{<:}\bar{\mathtt{Q}} \vdash \mathtt{S}_0 \text{ ok.}$$

On the other hand, since $\Delta \vdash \mathtt{N}$ ok, we have $\Delta \vdash \bar{\mathtt{T}}$ ok and $\Delta \vdash \bar{\mathtt{T}} \texttt{<:} [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$ by the rule WF-CLASS. Then, by Lemmas A.2.1 and A.2.6,

$$\Delta, \bar{\mathtt{Y}}\texttt{<:}[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{Q}} \vdash [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{S}_0 \text{ ok,}$$

finishing the case.

Case MT-SUPER. Since $\mathit{CT}(\mathtt{C})$is ok, by the rule GT-CLASS, $\bar{\mathtt{X}}\texttt{<:}\bar{\mathtt{N}} \vdash \mathtt{P}$ ok. By Lemmas A.2.1 and A.2.6, $\Delta \vdash [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{P}$ ok. The induction hypothesis finishes the case. □

LEMMA A.3.4. *If* $\Delta \vdash \Gamma$ ok *and* $\Delta; \Gamma \vdash_{\mathrm{FGJ}} \mathtt{e}\texttt{:}\mathtt{T}$ *for some well-formed type environment* Δ*, then* $\Delta \vdash \mathtt{T}$ ok.

PROOF. By induction on the derivation of $\Delta; \Gamma \vdash_{\mathrm{FGJ}} \mathtt{e}\texttt{:}\mathtt{T}$ with a case analysis on the last rule used.

Case GT-VAR. Immediate from the definition of the well-formedness of Γ.

Case GT-FIELD. $\Delta; \Gamma \vdash_{\mathrm{FGJ}} \mathtt{e}_0\texttt{:}\mathtt{T}_0$ $\quad \mathit{fields}_{\mathrm{FGJ}}(\mathit{bound}_\Delta(\mathtt{T}_0)) = \bar{\mathtt{T}}\ \bar{\mathtt{f}}$

By the induction hypothesis, $\Delta \vdash \mathtt{T}_0$ ok. Since Δ is well formed, $\Delta \vdash$ *bound*$_\Delta(\mathtt{T}_0)$ ok. Then, by Lemma A.3.2, we have $\Delta \vdash \bar{\mathtt{T}}$ ok, finishing the case.

Case GT-INVK. $\Delta; \Gamma \vdash_{\mathrm{FGJ}} \mathtt{e}_0 : \mathtt{T}_0$
$mtype_{\mathrm{FGJ}}(\mathtt{m}, bound_\Delta(\mathtt{T}_0)) = \texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\texttt{>}\bar{\mathtt{U}} \to \mathtt{U}_0$
$\Delta \vdash \bar{\mathtt{V}}$ ok $\quad \Delta \vdash \bar{\mathtt{V}} <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}}$
$\Delta; \Gamma \vdash_{\mathrm{FGJ}} \bar{\mathtt{e}} : \bar{\mathtt{S}} \quad \Delta \vdash \bar{\mathtt{S}} <: [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}} \quad \mathtt{T} = [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}_0$

By the induction hypothesis, $\Delta \vdash \mathtt{T}_0$ ok. Since Δ is well formed, $\Delta \vdash$ *bound*$_\Delta(\mathtt{T}_0)$ ok. Then, by Lemma A.3.3, $\Delta, \bar{\mathtt{Y}} <: \bar{\mathtt{P}} \vdash \mathtt{U}_0$ ok. Finally, by Lemma A.2.6, we have $\Delta \vdash [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}_0$ ok, finishing the case.

Case GT-UCAST. $\Delta; \Gamma \vdash_{\mathrm{FGJ}} \mathtt{e}_0 : \mathtt{T}_0 \quad \Delta \vdash \mathtt{T}_0 <: \mathtt{N} \quad \mathtt{N} = \mathtt{T}$

By the induction hypothesis, $\Delta \vdash \mathtt{T}_0$ ok. By Lemma A.3.1, $\Delta \vdash \mathtt{N}$ ok, finishing the case.

Case GT-NEW, GT-DCAST, GT-SCAST. Immediate, from the fact that $\mathtt{T}$ is well formed by a premise of the rules. □

After developing several lemmas about erasure, we prove Theorem 4.5.1. Note that in the following discussions the erased class table $|CT|$ is *not* assumed to be ok; even so, however, if CT is ok, then the erased class table $|CT|$ itself is well defined, and thus *fields*$_{\mathrm{FJ}}$, *mtype*$_{\mathrm{FJ}}$, *mbody*$_{\mathrm{FJ}}$, and $<:_{\mathrm{FJ}}$ can be defined from $|CT|$.

LEMMA A.3.5. *If* $\Delta \vdash \mathtt{S} <:_{\mathrm{FGJ}} \mathtt{T}$*, then* $|\mathtt{S}|_\Delta <:_{\mathrm{FJ}} |\mathtt{T}|_\Delta$.

PROOF. Straightforward induction on the derivation of $\Delta \vdash \mathtt{S} <:_{\mathrm{FGJ}} \mathtt{T}$. □

LEMMA A.3.6. *If* $\Delta_1, \bar{\mathtt{X}} <: \bar{\mathtt{N}}, \Delta_2 \vdash \mathtt{U}$ ok *where none of* $\bar{\mathtt{X}}$ *appear in* Δ_1*, and* $\Delta_1 \vdash \bar{\mathtt{T}} <:_{\mathrm{FGJ}} [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$*, then* $|[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{U}|_{\Delta_1, [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\Delta_2} <:_{\mathrm{FJ}} |\mathtt{U}|_\Delta$.

PROOF. If $\mathtt{U}$ is nonvariable or a type variable $\mathtt{Y} \not\in \bar{\mathtt{X}}$, then the result is trivial. If $\mathtt{U}$ is a type variable $\mathtt{X}_i$, it is also easy, since $[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{U} = \mathtt{T}_i$ and, by Lemma A.3.5, $|\mathtt{T}_i|_{\Delta_1, [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\Delta_2} = |\mathtt{T}_i|_{\Delta_1} <:_{\mathrm{FJ}} |[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{N}_i|_{\Delta_1} = |\mathtt{N}_i|_\Delta = |\mathtt{X}|_\Delta$. □

LEMMA A.3.7. *If* $\Delta \vdash \mathtt{C}\texttt{<}\bar{\mathtt{U}}\texttt{>}$ ok *and* $fields_{\mathrm{FGJ}}(\mathtt{C}\texttt{<}\bar{\mathtt{U}}\texttt{>}) = \bar{\mathtt{V}}\ \bar{\mathtt{f}}$*, then* $fieldsmax(\mathtt{C}) = \bar{\mathtt{D}}\ \bar{\mathtt{f}}$ *and* $|\bar{\mathtt{V}}|_\Delta <:_{\mathrm{FGJ}} \bar{\mathtt{D}}$.

PROOF. By induction on the derivation of *fields*$_{\mathrm{FGJ}}(\mathtt{C}\texttt{<}\bar{\mathtt{U}}\texttt{>})$ using Lemma A.3.6 and the fact that $\Delta \vdash \bar{\mathtt{U}} <: [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$, where `class` $\mathtt{C}\texttt{<}\bar{\mathtt{X}} \triangleleft \bar{\mathtt{N}}\texttt{>}$ ..., derived from the rule WF-CLASS. □

LEMMA A.3.8. *If* $\Delta \vdash \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$ ok *and* $mtype_{\mathrm{FGJ}}(\mathtt{m}, \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}) = \texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\texttt{>}\bar{\mathtt{U}} \to \mathtt{U}_0$ *where* $\Delta \vdash \bar{\mathtt{V}} <:_{\mathrm{FGJ}} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}}$*, then* $mtypemax(\mathtt{m}, \mathtt{C}) = \bar{\mathtt{C}} \to \mathtt{C}_0$ *and* $|[\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}}|_\Delta <:_{\mathrm{FJ}} \bar{\mathtt{C}}$ *and* $|[\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}_0|_\Delta <:_{\mathrm{FJ}} \mathtt{C}_0$.

PROOF. Since $\Delta \vdash \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$ ok, we can have a sequence of type $\bar{\mathtt{S}}$ such that $\mathtt{S}_1 = \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}$ and $\mathtt{S}_n = \mathtt{Object}$ and $\Delta \vdash \mathtt{S}_i <:_{\mathrm{FGJ}} \mathtt{S}_{i+1}$ derived by the rule S-CLASS for any i. We prove by induction on the length n (≥ 2) of the sequence.

Case. $n = 2$. It must be the case that
$\texttt{class C<}\bar{\texttt{X}} \triangleleft \bar{\texttt{N}}\texttt{>} \triangleleft \texttt{Object \{ ...}$
$\texttt{<}\bar{\texttt{Y}} \triangleleft \bar{\texttt{Q}}\texttt{>W}_0\ \texttt{m (}\bar{\texttt{W}}\ \bar{\texttt{x}}\texttt{) \{...\} ...\}}$.

By the definition of *mtypemax*, $\bar{\texttt{C}} = |\bar{\texttt{W}}|_{\bar{\texttt{X}}<:\bar{\texttt{N}},\bar{\texttt{Y}}<:\bar{\texttt{Q}}}$ and $\texttt{C}_0 = |\texttt{W}_0|_{\bar{\texttt{X}}<:\bar{\texttt{N}},\bar{\texttt{Y}}<:\bar{\texttt{Q}}}$. Without loss of generality, we can assume $\bar{\texttt{X}}$ and $\bar{\texttt{Y}}$ are distinct. By the definition of $mtype_{\text{FGJ}}$,

$$[\bar{\texttt{T}}/\bar{\texttt{X}}]\bar{\texttt{Q}} = \bar{\texttt{P}}$$
$$[\bar{\texttt{T}}/\bar{\texttt{X}}]\bar{\texttt{W}} = \bar{\texttt{U}}$$
$$[\bar{\texttt{T}}/\bar{\texttt{X}}]\texttt{W}_0 = \texttt{U}_0,$$

and therefore

$$\Delta \vdash \bar{\texttt{V}} <:_{\text{FGJ}} [\bar{\texttt{V}}/\bar{\texttt{Y}}][\bar{\texttt{T}}/\bar{\texttt{X}}]\bar{\texttt{Q}}.$$

Moreover, by the rule WF-Class, we have

$$\Delta \vdash \bar{\texttt{T}} <: [\bar{\texttt{T}}/\bar{\texttt{X}}]\bar{\texttt{N}} \quad (= [\bar{\texttt{V}}/\bar{\texttt{Y}}][\bar{\texttt{T}}/\bar{\texttt{X}}]\bar{\texttt{N}},\ \text{since } \bar{\texttt{Y}} \text{ does not appear in } [\bar{\texttt{T}}/\bar{\texttt{X}}]\bar{\texttt{N}}).$$

By Lemma A.3.6, $|[\bar{\texttt{V}}/\bar{\texttt{Y}}][\bar{\texttt{T}}/\bar{\texttt{X}}]\bar{\texttt{W}}|_\Delta <:_{\text{FJ}} \text{C}$ and $|[\bar{\texttt{V}}/\bar{\texttt{Y}}][\bar{\texttt{T}}/\bar{\texttt{X}}]\texttt{W}_0|_\Delta <:_{\text{FJ}} \text{C}_0$, finishing the case.

Case. $n = k + 1$. Suppose
$\texttt{class C<}\bar{\texttt{X}} \triangleleft \bar{\texttt{N}}\texttt{>} \triangleleft \texttt{N \{...\}}$.

Note that $\Delta \vdash \texttt{C<}\bar{\texttt{T}}\texttt{>} <:_{\text{FGJ}} [\bar{\texttt{T}}/\bar{\texttt{X}}]\texttt{N}$ by the rule S-Class. Now, we have three subcases:

Subcase. $mtype_{\text{FGJ}}(\texttt{m}, [\bar{\texttt{T}}/\bar{\texttt{X}}]\texttt{N})$ is undefined. The method `m` must be declared in `C`. Similarly for the base case above.

Subcase. $mtype_{\text{FGJ}}(\texttt{m}, [\bar{\texttt{T}}/\bar{\texttt{X}}]\texttt{N})$ is well defined, and `m` is defined in `C`. By the rule GT-Method, it must be the case that

$$mtype_{\text{FGJ}}(\texttt{m}, [\bar{\texttt{T}}/\bar{\texttt{X}}]\texttt{N}) = \texttt{<}\bar{\texttt{Y}} \triangleleft \bar{\texttt{P}}\texttt{>}\bar{\texttt{U}} \rightarrow \texttt{U}_0{}'$$

where $\Delta, \bar{\texttt{Y}}<:\bar{\texttt{P}} \vdash \texttt{U}_0 <:_{\text{FGJ}} \texttt{U}_0{}'$. By Lemmas A.2.5 and A.3.5, $|[\bar{\texttt{V}}/\bar{\texttt{Y}}]\texttt{U}_0|_\Delta <:_{\text{FJ}} |[\bar{\texttt{V}}/\bar{\texttt{Y}}]\texttt{U}_0{}'|_\Delta$. The induction hypothesis and transitivity of $<:_{\text{FJ}}$ finish the subcase.

Subcase. $mtype_{\text{FGJ}}(\texttt{m}, [\bar{\texttt{T}}/\bar{\texttt{X}}]\texttt{N})$ is well defined, and `m` is not defined in `C`. It is easy because $mtype_{\text{FGJ}}(\texttt{m}, [\bar{\texttt{T}}/\bar{\texttt{X}}]\texttt{N}) = mtype_{\text{FGJ}}(\texttt{m}, \texttt{C<}\bar{\texttt{T}}\texttt{>})$ by the rule MT-Super. The induction hypothesis finishes the subcase. □

Proof of Theorem 4.5.1. We prove the theorem in two steps: first, it is shown that if $\Delta; \Gamma \vdash_{\text{FGJ}} \texttt{e}:\texttt{T}$ then $|\Gamma|_\Delta \vdash_{\text{FJ}} |\texttt{e}|_{\Delta,\Gamma} : |\texttt{T}|_\Delta$; and second, we show $|CT|$ is ok.

The first part is proved by induction on the derivation of $\Delta; \Gamma \vdash_{\text{FGJ}} \texttt{e}:\texttt{T}$ with a case analysis on the last rule used.

Case GT-Field. $\texttt{e} = \texttt{e}_0\texttt{.f}_i$ $\quad \Delta; \Gamma \vdash_{\text{FGJ}} \texttt{e}_0 : \texttt{T}_0$
$fields_{\text{FGJ}}(bound_\Delta(\texttt{T}_0)) = \bar{\texttt{T}}\ \bar{\texttt{f}} \quad \texttt{T} = \texttt{T}_i$

By the induction hypothesis, we have $|\Gamma|_\Delta \vdash_{\text{FJ}} |\texttt{e}_0|_\Delta : |\texttt{T}_0|_\Delta$. By Lemma A.3.4, $\Delta \vdash \texttt{T}_0$ ok. Then, whether $\texttt{T}_0$ is a type variable or not, we have by Lemma A.3.7 $fieldsmax(|\texttt{T}_0|_\Delta) = \bar{\texttt{C}}\ \bar{\texttt{f}}$ and $|\bar{\texttt{T}}|_\Delta <: \bar{\texttt{C}}$. Note that by definition it is obvious that $fields_{\text{FJ}}(\texttt{C}) = fieldsmax(\texttt{C})$. By the rule T-Field, we have $|\Gamma|_\Delta \vdash_{\text{FJ}} |\texttt{e}_0|_{\Delta,\Gamma}\texttt{.f}_i : \texttt{C}_i$.

If $|\mathtt{T}_i|_\Delta = \mathtt{C}_i$, then the equation $|\mathtt{e}_0.\mathtt{f}_i|_{\Delta,\Gamma} = |\mathtt{e}_0|_{\Delta,\Gamma}.\mathtt{f}_i$ derived from the rule E-FIELD finishes the case. On the other hand, if $|\mathtt{T}_i|_\Delta \neq \mathtt{C}_i$, then

$$|\mathtt{e}_0.\mathtt{f}_i|_{\Delta,\Gamma} = (|\mathtt{T}_i|_\Delta)^s|\mathtt{e}_0|_{\Delta,\Gamma}.\mathtt{f}_i$$

by the rule E-FIELD-CAST and $|\Gamma|_\Delta \vdash_{FJ} (|\mathtt{T}|_\Delta)^s|\mathtt{e}_0|_{\Delta,\Gamma}.\mathtt{f}_i : |\mathtt{T}|_\Delta$ by the rule T-DCAST, finishing the case. Note that the synthetic cast is not stupid.

Case GT-INVK. Similar to the case above.

Case GT-New, GT-UCAST, GT-DCAST, GT-SCAST. Easy. Notice that the nature of the cast (up, down, or stupid) is also preserved.

The second part ($|CT|$ is ok) follows from the first part with examination of the rules GT-METHOD and GT-CLASS. We show that if $\mathtt{M}$ OK IN $\mathtt{C}\texttt{<}\bar{\mathtt{X}} \triangleleft \bar{\mathtt{N}}\texttt{>}$, then $|\mathtt{M}|_{\bar{\mathtt{X}}<:\bar{\mathtt{N}},\mathtt{C}}$ OK IN C. Suppose

$$\begin{array}{l}
\mathtt{M} = \texttt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\texttt{>}\ \mathtt{T}\ \mathtt{m}(\bar{\mathtt{T}}\ \mathtt{x})\{\ \mathtt{return}\ \mathtt{e}_0;\ \}\\
|\mathtt{M}|_{\bar{\mathtt{X}}<:\bar{\mathtt{N}},\mathtt{C}} = \mathtt{D}\ \mathtt{m}(\bar{\mathtt{D}}\ \mathtt{x}')\{\ \mathtt{return}\ \mathtt{e}_0';\ \}\\
\mathit{mtypemax}(\mathtt{m},\mathtt{C}) = \bar{\mathtt{D}} \to \mathtt{D}\\
\Gamma = \bar{\mathtt{x}} : \bar{\mathtt{T}}\\
\Delta = \bar{\mathtt{X}}<:\bar{\mathtt{N}}, \bar{\mathtt{Y}}<:\bar{\mathtt{P}}\\
\mathtt{e}_i = \begin{cases} \mathtt{x}_i' & \text{if } \mathtt{D}_i = |\mathtt{T}_i|_\Delta\\ (|\mathtt{T}_i|_\Delta)^s\mathtt{x}_i' & \text{otherwise}\end{cases}\\
\mathtt{e}_0' = [\bar{\mathtt{e}}/\bar{\mathtt{x}}](|\mathtt{e}_0|_{\Delta,(\Gamma,\mathtt{this}:\mathtt{C}<\bar{\mathtt{X}}>)}).
\end{array}$$

By the rule GT-METHOD, we have

$$\begin{array}{l}
\Delta \vdash \bar{\mathtt{T}}, \mathtt{T}, \bar{\mathtt{P}}\ \text{ok}\\
\Delta; \Gamma, \mathtt{this} : \mathtt{C}\texttt{<}\bar{\mathtt{X}}\texttt{>} \vdash_{FGJ} \mathtt{e}_0 : \mathtt{S}\\
\Delta \vdash \mathtt{S} <:_{FGJ} \mathtt{T}\\
\text{if } \mathit{mtype}_{FGJ}(\mathtt{m},\mathtt{N}) = \texttt{<}\bar{\mathtt{Z}} \triangleleft \bar{\mathtt{Q}}\texttt{>}\bar{\mathtt{U}}\to\mathtt{U}, \text{ then } \bar{\mathtt{P}}, \bar{\mathtt{T}} = [\bar{\mathtt{Y}}/\bar{\mathtt{Z}}](\bar{\mathtt{Q}}, \bar{\mathtt{U}}) \text{ and } \Delta \vdash \mathtt{T} <:_{FGJ} [\bar{\mathtt{Y}}/\bar{\mathtt{Z}}]\mathtt{U}
\end{array}$$

where class $\mathtt{C}\texttt{<}\bar{\mathtt{X}} \triangleleft \bar{\mathtt{N}}\texttt{>} \triangleleft \mathtt{N}\ \{\cdots\}$. We must show that

$$\begin{array}{l}
\bar{\mathtt{x}}' : \mathtt{D}, \mathtt{this} : \mathtt{C} \vdash_{FJ} \mathtt{e}' : \mathtt{E}\\
\mathtt{E} <:_{FJ} \mathtt{D}\\
\text{if } \mathit{mtype}_{FJ}(\mathtt{m}, |\mathtt{N}|_\Delta) = \bar{\mathtt{E}}\to\mathtt{D}', \text{ then } \bar{\mathtt{E}} = \bar{\mathtt{D}} \text{ and } \mathtt{D}' = \mathtt{D}
\end{array}$$

for some E. By the result of the first part, $|\Gamma|_\Delta, \mathtt{this} : \mathtt{C} \vdash_{FJ} |\mathtt{e}|_{\Delta,\Gamma} : |\mathtt{S}|_\Delta$. Since, by Lemma A.3.8, $|\mathtt{T}_i|_\Delta <: \mathtt{D}_i$, we have $\mathtt{x}_i' : \mathtt{D}_i \vdash \mathtt{e}_i : |\mathtt{T}_i|_\Delta$. By Lemma A.2.11,

$$\bar{\mathtt{x}}' : \bar{\mathtt{D}}, \mathtt{this} : \mathtt{C} \vdash \mathtt{e}_0' : \mathtt{C}_0$$

for some $\mathtt{C}_0$ where $\mathtt{C}_0 <:_{FJ} |\mathtt{S}|_\Delta$. On the other hand, by Lemma A.3.8, $|\mathtt{T}|_\Delta <:_{FJ} \mathtt{D}$. Since we have $|\mathtt{S}|_\Delta <:_{FJ} |\mathtt{T}|_\Delta$ by Lemma A.3.5, $\mathtt{C}_0 <:_{FJ} \mathtt{D}$ by transitivity of $<:$. Let E be $\mathtt{C}_0$. Finally, suppose $\mathit{mtypemax}(\mathtt{m}, |\mathtt{N}|_\Delta)$ is well defined. Then, $\mathit{mtype}_{FGJ}(\mathtt{m},\mathtt{N})$ is also well defined. By definition, $\mathit{mtypemax}(\mathtt{m}, |\mathtt{N}|_\Delta) = \bar{\mathtt{D}} \to \mathtt{D} = \mathit{mtype}_{FJ}(\mathtt{m}, |\mathtt{N}|_\Delta)$.

It is easy to show that L OK in FGJ implies |L| OK in FJ. □

A.4 Proof of Theorems 4.5.3 and 4.5.4

In the rest of this section, we prove these theorems and corollaries; we first prove the required lemmas.

Lemma A.4.1. *If* $\Gamma, \bar{\mathtt{x}}\!:\!\bar{\mathtt{B}} \vdash \mathtt{e} \overset{\mathrm{exp}}{\Rightarrow} \mathtt{e}'$ *and* $\Gamma \vdash_{\mathrm{FJ}} \bar{\mathtt{d}}\!:\!\bar{\mathtt{A}}$ *where* $\bar{\mathtt{A}} \mathrel{<:}_{\mathrm{FJ}} \bar{\mathtt{B}}$, *then* $\Gamma \vdash [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e} \overset{\mathrm{exp}}{\Rightarrow} [\bar{\mathtt{d}}/\bar{\mathtt{x}}]\mathtt{e}'$.

PROOF. By induction on the derivation of $\Gamma, \bar{\mathtt{x}}\!:\!\bar{\mathtt{B}} \vdash_{\mathrm{FJ}} \mathtt{e}\!:\!\mathtt{C}$. □

Lemma A.4.2. *Suppose* $dom(\Gamma) = dom(\Gamma')$ *and* $\Delta = \Delta_1, \bar{\mathtt{X}}\mathrel{<:}\bar{\mathtt{N}}, \Delta_2$ *where none of* $\bar{\mathtt{X}}$ *appears in* Δ_1. *If* $\Delta;\Gamma \vdash_{\mathrm{FGJ}} \mathtt{e}\!:\!\mathtt{T}$ *and* $\Delta_1 \vdash \bar{\mathtt{U}} \mathrel{<:}_{\mathrm{FGJ}} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$ *where* $\Delta_1 \vdash \bar{\mathtt{U}}$ ok, *and* $\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \Gamma'(\mathtt{x}) \mathrel{<:}_{\mathrm{FGJ}} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Gamma(\mathtt{x})$ *for all* $\mathtt{x} \in dom(\Gamma)$, *then* $|\mathtt{e}|_{\Delta,\Gamma}$ *is obtained from* $|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'}$ *by some combination of replacements of some synthetic casts* $(\mathtt{D})^s$ *with* $(\mathtt{C})^s$ *where* $\mathtt{D} \mathrel{<:} \mathtt{C}$, *or removals of some synthetic casts.*

PROOF. By induction on the derivation of $\Delta;\Gamma \vdash \mathtt{e}\!:\!\mathtt{T}$ with a case analysis on the last rule used.

Case GT-VAR. Trivial.
Case GT-FIELD. $\mathtt{e} = \mathtt{e}_0.\mathtt{f}$ $\qquad \Delta;\Gamma \vdash \mathtt{e}_0\!:\!\mathtt{T}_0$
$fields_{\mathrm{FGJ}}(bound_\Delta(\mathtt{T}_0)) = \bar{\mathtt{T}}\ \bar{\mathtt{f}} \qquad \mathtt{T} = \mathtt{T}_i$

By the induction hypothesis, $|\mathtt{e}_0|_{\Delta,\Gamma}$ is obtained from $|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}_0|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'}$ by some combination of replacements of some synthetic casts $(\mathtt{D})^s$ with $(\mathtt{C})^s$ where $\mathtt{D}\mathrel{<:}_{\mathrm{FJ}}\mathtt{C}$, or removals of some synthetic casts. By Theorem 4.5.1, $|\Gamma|_\Delta \vdash_{\mathrm{FJ}} |\mathtt{e}_0|_{\Delta,\Gamma}\!:\!|\mathtt{T}_0|_\Delta$. By Lemma A.3.7, $fieldsmax(|\mathtt{T}_0|_\Delta) = \bar{\mathtt{C}}\ \bar{\mathtt{f}}$ and $|\bar{\mathtt{T}}|_\Delta \mathrel{<:}_{\mathrm{FJ}} \bar{\mathtt{C}}$.

We now have two subcases.

Subcase. $|\mathtt{T}_i|_\Delta \neq \mathtt{C}_i$

By the rule E-FIELD-CAST,

$$|\mathtt{e}|_{\Delta,\Gamma} = (|\mathtt{T}_i|_\Delta)^s|\mathtt{e}_0|_{\Delta,\Gamma}.\mathtt{f}_i.$$

Now we must show that $|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'} = (\mathtt{D})^s|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}_0|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'}.\mathtt{f}_i$ for some $\mathtt{D} \mathrel{<:}_{\mathrm{FJ}} |\mathtt{T}|_\Delta$. By Lemmas A.2.10 and A.2.11,

$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2;\Gamma' \vdash_{\mathrm{FGJ}} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}_0\!:\!\mathtt{S}_0$$
$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \mathtt{S}_0 \mathrel{<:}_{\mathrm{FGJ}} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_0.$$

By Lemmas A.2.7 and A.2.8,

$$fields_{\mathrm{FGJ}}(bound_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}(\mathtt{S}_0)) = ([\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{T}}\ \bar{\mathtt{f}}), \bar{\mathtt{T}}'\ \bar{\mathtt{g}}.$$

Then, by Lemma A.3.6,

$$|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_i|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2} \mathrel{<:}_{\mathrm{FJ}} |\mathtt{T}_i|_\Delta.$$

On the other hand,

$$fieldsmax(|\mathtt{S}_0|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}) = \bar{\mathtt{C}}\ \bar{\mathtt{f}}, \bar{\mathtt{D}}\ \bar{\mathtt{g}}$$

for some $\bar{\mathtt{D}}$. Therefore, by the rule E-FIELD-CAST,

$$|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'} = \left(|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_i|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}\right)^s|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'}.\mathtt{f}_i,$$

finishing the subcase.

Subcase. $|\mathtt{T}_i|_\Delta = \mathtt{C}_i$

Similar to the subcase above.

Case GT-Method. $\mathtt{e} = \mathtt{e}_0\mathtt{.m<}\bar{\mathtt{V}}\mathtt{>(}\bar{\mathtt{d}}\mathtt{)}$ $\quad \Delta;\Gamma \vdash_{FGJ} \mathtt{e}_0 : \mathtt{T}_0$
$\mathit{mtype}_{FGJ}(\mathtt{m}\mathit{bound}_\Delta(\mathtt{T}_0)) = \mathtt{<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}\mathtt{>}\bar{\mathtt{U}} \to \mathtt{U}_0$
$\Delta \vdash \bar{\mathtt{V}}$ ok $\quad \Delta \vdash \bar{\mathtt{V}} <:_{FGJ} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}}$
$\Delta;\Gamma \vdash_{FGJ} \bar{\mathtt{d}} : \bar{\mathtt{S}} \quad \Delta \vdash \bar{\mathtt{S}} <:_{FGJ} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}}$
$\mathtt{T} = [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}_0$

By the induction hypothesis, $|\bar{\mathtt{d}}|_{\Delta,\Gamma}$ are obtained from $|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{d}}|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'}$ by some combination of replacements of some synthetic casts $(\mathtt{D})^s$ with $(\mathtt{C})^s$ where $\mathtt{D} <:_{FJ} \mathtt{C}$, or removals of some synthetic casts. By Theorem 4.5.1, $|\Gamma|_\Delta \vdash_{FJ} |\mathtt{e}_0|_{\Delta,\Gamma} : |\mathtt{T}_0|_\Delta$. By Lemma A.3.8, $\mathit{mtypemax}(\mathtt{m}, |\mathtt{T}_0|_\Delta) = \bar{\mathtt{E}} \to \mathtt{E}_0$ and $|\mathtt{T}|_\Delta <:_{FJ} \mathtt{E}_0$.

We now have two subcases:

Subcase. $|\mathtt{T}|_\Delta \neq \mathtt{E}_0$

By the rule E-Invk-Cast,

$$|\mathtt{e}|_{\Delta,\Gamma} = (|\mathtt{T}|_\Delta)^s|\mathtt{e}_0|_{\Delta,\Gamma}\mathtt{.m(}|\bar{\mathtt{d}}|_{\Delta,\Gamma}\mathtt{)}.$$

Now, we must show that

$$|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'} = (\mathtt{D})^s|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}_0|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'}\mathtt{.m(}|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{d}}|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'}\mathtt{)}$$

for some $\mathtt{D} <:_{FJ} |\mathtt{T}|_\Delta$. By Lemmas A.2.10 and A.2.11,

$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2; \Gamma' \vdash_{FGJ} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}_0 : \mathtt{S}_0$$
$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash \mathtt{S}_0 <:_{FGJ} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_0.$$

Without loss of generality, we can assume $\bar{\mathtt{X}}$ and $\bar{\mathtt{Y}}$ are distinct. By Lemmas A.2.7 and A.2.9, we have

$$\mathit{mtype}_{FGJ}(\mathtt{m}, \mathit{bound}_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}(\mathtt{S}_0)) = \mathtt{<}\bar{\mathtt{Y}} \triangleleft [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{P}}\mathtt{>}[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{U}} \to \mathtt{U}_0'$$
$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2, \bar{\mathtt{Y}} <: [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{P}} \vdash \mathtt{U}_0' <:_{FGJ} [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{U}_0.$$

By Lemma A.2.5,

$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{V}} <:_{FGJ} [\bar{\mathtt{U}}/\bar{\mathtt{X}}][\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}} \quad (= [[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{V}}/\bar{\mathtt{Y}}]([\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{P}}))$$

and by the same lemma,

$$\Delta_1, [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2 \vdash [[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}_0' <:_{FGJ} [[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{V}}/\bar{\mathtt{Y}}][\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{U}_0 \quad (= [\bar{\mathtt{U}}/\bar{\mathtt{X}}][\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}_0 = [\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}).$$

Then, by Lemmas A.3.5 and A.3.6,

$$|[[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}_0'|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2} <:_{FJ} |[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2} <:_{FJ} |\mathtt{T}|_\Delta.$$

On the other hand, it is easy to show that

$$\mathit{mtypemax}(\mathtt{m}, |\mathtt{S}_0|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}) = \mathit{mtypemax}(\mathtt{m}, |[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{T}_0|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}) = \bar{\mathtt{E}} \to \mathtt{E}_0.$$

Then, by the rule E-Invk-Cast,

$$\begin{aligned}&|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'}\\ &\quad = \left(|[[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}_0'|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2}\right)^s|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\mathtt{e}_0|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'}\mathtt{.m}\left(|[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\bar{\mathtt{d}}|_{\Delta_1,[\bar{\mathtt{U}}/\bar{\mathtt{X}}]\Delta_2,\Gamma'}\right),\end{aligned}$$

finishing the subcase.

Subcase. $|\mathtt{T}|_{\Delta,\Gamma} = \mathtt{E}_0$

Similar to the subcase above.

Case GT-NEW, GT-UCAST, GT-DCAST, GT-SCAST. Immediate from the induction hypothesis. □

LEMMA A.4.3. *Suppose*

(1) $mbody_{\mathrm{FGJ}}(\mathtt{m}{<}\bar{\mathtt{V}}{>}, \mathtt{C}{<}\bar{\mathtt{T}}{>}) = \bar{\mathtt{x}}.\mathtt{e}$,
(2) $mtype_{\mathrm{FGJ}}(\mathtt{m}, \mathtt{C}{<}\bar{\mathtt{T}}{>}) = {<}\bar{\mathtt{Y}} \triangleleft \bar{\mathtt{P}}{>}\bar{\mathtt{U}} \to \mathtt{U}_0$,
(3) $\Delta \vdash \mathtt{C}{<}\bar{\mathtt{T}}{>}$ ok,
(4) $\Delta \vdash \bar{\mathtt{V}} <:_{\mathrm{FGJ}} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{P}}$,
(5) $\Delta \vdash \bar{\mathtt{W}} <:_{\mathrm{FGJ}} [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}}$, *and*
(6) $mbody_{\mathrm{FJ}}(\mathtt{m}, \mathtt{C}) = \bar{\mathtt{x}}.\mathtt{e}'$.

Then, $|\bar{\mathtt{x}} : \bar{\mathtt{W}}, \mathtt{this} : \mathtt{C}{<}\bar{\mathtt{T}}{>}|_\Delta \vdash |\mathtt{e}|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{C}<\bar{\mathtt{T}}>} \overset{\mathrm{exp}}{\Rightarrow} \mathtt{e}'$.

PROOF. By induction on the derivation of $mbody_{\mathrm{FGJ}}(\mathtt{m}{<}\bar{\mathtt{V}}{>}, \mathtt{C}{<}\bar{\mathtt{T}}{>})$, with a case analysis on the last rule used.

Case MB-Class.
```
class C<X̄ ◁ N̄> ◁ N { ...
  <Ȳ ◁ Q̄> S₀ m(S̄ x){ return e₀;}}
```
$[\bar{\mathtt{T}}/\bar{\mathtt{X}}, \bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{e}_0 = \mathtt{e}$
$[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{Q}} = \bar{\mathtt{P}}$
$[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{S}} = \bar{\mathtt{U}}$
$[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{S}_0 = \mathtt{U}_0$

Let $\Delta' = \bar{\mathtt{X}}{<:}\bar{\mathtt{N}}, \bar{\mathtt{Y}}{<:}\bar{\mathtt{Q}}$ and $\Gamma = \bar{\mathtt{x}} : \bar{\mathtt{S}}, \mathtt{this} : \mathtt{C}{<}\bar{\mathtt{X}}{>}$. By the rule WF-CLASS, $\Delta \vdash \bar{\mathtt{T}} <:_{\mathrm{FGJ}} [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{N}}$ $(= [\bar{\mathtt{V}}/\bar{\mathtt{Y}}][\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{N}})$. By Lemma A.4.2, $|\mathtt{e}_0|_{\Delta',\bar{\mathtt{x}}:\bar{\mathtt{S}},\mathtt{this}:\mathtt{C}<\bar{\mathtt{X}}>}$ is obtained from $|\mathtt{e}|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{C}<\bar{\mathtt{T}}>}$ by some combination of replacements of some synthetic casts $(\mathtt{B})^s$ with $(\mathtt{A})^s$ where $\mathtt{B} <:_{\mathrm{FJ}} \mathtt{A}$, or removals of some synthetic casts. By Theorem 4.5.1,

$$|\bar{\mathtt{x}} : \bar{\mathtt{S}}, \mathtt{this} : \mathtt{C}{<}\bar{\mathtt{X}}{>}|_{\Delta'} \vdash_{\mathrm{FJ}} |\mathtt{e}_0|_{\Delta',\bar{\mathtt{x}}:\bar{\mathtt{S}},\mathtt{this}:\mathtt{C}<\bar{\mathtt{X}}>} : |\mathtt{S}_0|_{\Delta'}.$$

Now, let $mtypemax(\mathtt{m}, \mathtt{C}) = \bar{\mathtt{D}} \to \mathtt{D}$ and

$$\mathtt{e}_i = \begin{cases} \mathtt{x}_i & \text{if } \mathtt{D}_i = |\mathtt{S}_i|_{\Delta'} \\ (|\mathtt{S}_i|_{\Delta'})^s \mathtt{x}_i & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, \#(\bar{\mathtt{x}})$. Since $\mathtt{e}' = [\bar{\mathtt{e}}/\bar{\mathtt{x}}]|\mathtt{e}_0|_{\Delta',\Gamma}$ and $|\bar{\mathtt{W}}|_\Delta <:_{\mathrm{FJ}} |[\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\bar{\mathtt{U}}|_\Delta <:_{\mathrm{FJ}} |\bar{\mathtt{S}}|_{\Delta'}$, by Lemmas A.3.5 and A.3.6, each $\mathtt{e}_i$ is either a variable or a variable with an upcast under the environment $|\bar{\mathtt{x}} : \bar{\mathtt{W}}, \mathtt{this} : \mathtt{C}{<}\bar{\mathtt{T}}{>}|_\Delta$. Then, we have

$$|\bar{\mathtt{x}} : \bar{\mathtt{W}}, \mathtt{this} : \mathtt{C}{<}\bar{\mathtt{T}}{>}|_\Delta \vdash_{\mathrm{FJ}} \mathtt{e}' : \mathtt{D}$$

for some $\mathtt{D}$ such that $\mathtt{D} <:_{\mathrm{FJ}} |\mathtt{S}_0|_{\Delta'}$ by Lemma A.1.2. Therefore, we have

$$|\bar{\mathtt{x}} : \bar{\mathtt{W}}, \mathtt{this} : \mathtt{C}{<}\bar{\mathtt{T}}{>}|_\Delta \vdash |\mathtt{e}|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{C}<\bar{\mathtt{T}}>} \overset{\mathrm{exp}}{\Rightarrow} \mathtt{e}',$$

finishing the case.

Case MB-SUPER. `class C<X̄ ◁ N̄> ◁ D<S̄> { ... M̄ }` $\mathtt{m} \notin \bar{\mathtt{M}}$

By the induction hypothesis,

$$\begin{tikzcd}
\mathsf{e} \arrow[r, "\text{reduce (FGJ)}"] \arrow[dd, "\text{erase}"'] & \mathsf{e}' \arrow[d, dotted, "\text{erase}"] \\
& |\mathsf{e}'| \arrow[d, Rightarrow, dotted] \\
|\mathsf{e}| \arrow[r, dotted, "\text{reduce (FJ)}"'] & \mathsf{d}'
\end{tikzcd}$$

Fig. 12.

$$|\bar{\mathtt{x}} : \bar{\mathtt{W}}, \mathtt{this} : [\bar{\mathtt{T}}/\bar{\mathtt{X}}]\mathtt{D}\texttt{<}\bar{\mathtt{S}}\texttt{>}|_\Delta \vdash |\mathsf{e}|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{D}\texttt{<}[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{S}}\texttt{>}} \overset{\text{exp}}{\Rightarrow} \mathsf{e}'.$$

By Lemma A.4.1,

$$|\bar{\mathtt{x}} : \bar{\mathtt{W}}, \mathtt{this} : \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}|_\Delta \vdash |\mathsf{e}|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{D}\texttt{<}[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{S}}\texttt{>}} \overset{\text{exp}}{\Rightarrow} \mathsf{e}'.$$

Then, by Lemma A.4.2, $|\mathsf{e}|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{D}\texttt{<}[\bar{\mathtt{T}}/\bar{\mathtt{X}}]\bar{\mathtt{S}}\texttt{>}}$ is obtained from $|\mathsf{e}|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}}$ by some combination of replacements of some synthetic casts $(\mathtt{B})^s$ with $(\mathtt{A})^s$ where $\mathtt{B} <:_{\text{FJ}} \mathtt{A}$, or removals of some synthetic casts. On the other hand, by Lemma A.1.2,

$$|\bar{\mathtt{x}} : \bar{\mathtt{W}}, \mathtt{this} : \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}|_\Delta \vdash_{\text{FJ}} \mathsf{e}' : \mathtt{E}$$

for some $\mathtt{E}$. Therefore,

$$|\bar{\mathtt{x}} : \bar{\mathtt{W}}, \mathtt{this} : \mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}|_\Delta \vdash |\mathsf{e}|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{C}\texttt{<}\bar{\mathtt{T}}\texttt{>}} \overset{\text{exp}}{\Rightarrow} \mathsf{e}',$$

finishing the case. □

LEMMA A.4.4. *If* $\Delta;\Gamma \vdash_{\text{FGJ}} \mathsf{e} : \mathtt{T}$ *and* $\mathsf{e} \rightarrow_{\text{FGJ}} \mathsf{e}'$*, then there exists some FJ expression* d' *such that* $|\Gamma|_\Delta \vdash_{\text{FJ}} |\mathsf{e}'|_{\Delta,\Gamma} \overset{\text{exp}}{\Rightarrow} \mathsf{d}'$ *and* $|\mathsf{e}|_{\Delta,\Gamma} \rightarrow_{\text{FJ}} \mathsf{d}'$. *In other words, the diagram shown in Figure* 12 *commutes.*

PROOF. By induction on the derivation of $\mathsf{e} \rightarrow_{\text{FGJ}} \mathsf{e}'$ with a case analysis on the last reduction rule used. We show the main base cases.

Case GR-FIELD. $\mathsf{e} = \mathtt{new\ N}(\bar{\mathsf{e}})\mathtt{.f}_i$ $\quad$ $\mathit{fields}_{\text{FGJ}}(\mathtt{N}) = \bar{\mathtt{T}}\ \bar{\mathtt{f}}$ $\quad$ $\mathsf{e}' = \mathsf{e}_i$

We have two subcases depending on the last erasure rule used.

Subcase E-FIELD-CAST. $|\mathsf{e}|_{\Delta,\Gamma} = (\mathtt{D})^s(\mathtt{new\ C}(|\bar{\mathsf{e}}|_{\Delta,\Gamma})\mathtt{.f}_i)$

We have $|\mathtt{N}|_\Delta = \mathtt{C}$ by definition of erasure. Since $\mathit{fields}_{\text{FJ}}(\mathtt{C}) = \bar{\mathtt{C}}\ \bar{\mathtt{f}}$ for some $\bar{\mathtt{C}}$, we have $|\mathsf{e}|_{\Delta,\Gamma} \rightarrow_{\text{FJ}} (\mathtt{D})^s|\mathsf{e}_i|_{\Delta,\Gamma}$. On the other hand, by Theorem 3.4.1, $\Delta;\Gamma \vdash_{\text{FGJ}} \mathsf{e}_i : \mathtt{T}_i$ such that $\Delta \vdash \mathtt{T}_i <:_{\text{FGJ}} \mathtt{T}$. By Theorem 4.5.1, $|\mathtt{T}|_\Delta = \mathtt{D}$ and $|\Gamma|_\Delta \vdash_{\text{FJ}} |\mathsf{e}_i|_{\Delta,\Gamma} : |\mathtt{T}_i|_\Delta$. Since $|\mathtt{T}_i|_\Delta <:_{\text{FJ}} \mathtt{D}$ by Lemma A.3.5, $(\mathtt{D})^s|\mathsf{e}_i|_{\Delta,\Gamma}$ is obtained by adding an upcast to $|\mathsf{e}_i|_{\Delta,\Gamma}$.

Subcase E-FIELD. $|\mathsf{e}|_{\Delta,\Gamma} = \mathtt{new\ C}(|\bar{\mathsf{e}}|_{\Delta,\Gamma})\mathtt{.f}_i$

Follows from the induction hypothesis.

$$\begin{tikzcd} \mathtt{e} \arrow[r, "\text{reduce (FJ)}"] \arrow[d, Rightarrow] & \mathtt{e}' \arrow[d, Rightarrow, dotted] \\ \mathtt{d} \arrow[r, dotted, "\text{reduce (FJ)}"', "*"] & \mathtt{d}' \end{tikzcd}$$

Fig. 13.

Case GR-INVK. $\mathtt{e} = \mathtt{new\ C<\bar{T}>(\bar{e}).m<\bar{V}>(\bar{d})}$
$\mathit{mbody}_{\mathrm{FGJ}}(\mathtt{m<\bar{V}>}, \mathtt{C<\bar{T}>}) = \mathtt{\bar{x}.e_0}$
$\mathtt{e}' = [\mathtt{\bar{d}/\bar{x}}, \mathtt{new\ C<\bar{T}>(\bar{e})/this}]\mathtt{e_0}$

We have two subcases, depending on the last erasure rule used.

Subcase E-INVK-CAST. $|\mathtt{e}|_{\Delta,\Gamma} = \mathtt{(D)}^{s}\mathtt{(new\ C(}|\mathtt{\bar{e}}|_{\Delta,\Gamma}\mathtt{).m(}|\mathtt{\bar{d}}|_{\Delta,\Gamma}\mathtt{))}$

Since $\mathit{mbody}_{\mathrm{FGJ}}(\mathtt{m<\bar{V}>}, \mathtt{C<\bar{T}>})$ is well defined, $\mathit{mbody}_{\mathrm{FJ}}(\mathtt{m}, \mathtt{C})$ is also well defined. Let $\mathit{mbody}_{\mathrm{FJ}}(\mathtt{m}, \mathtt{C}) = \mathtt{\bar{x}.e_0}'$ and $\Gamma' = \mathtt{\bar{x}} : \mathtt{\bar{U}}, \mathtt{this} : \mathtt{C<\bar{T}>}$ where $\mathtt{\bar{U}}$ are types of $\mathtt{\bar{d}}$. Then, by Lemma A.4.3,

$$|\Gamma'|_{\Delta} \vdash |\mathtt{e_0}|_{\Delta,\Gamma'} \overset{\text{exp}}{\Longrightarrow} \mathtt{e_0}'.$$

By Lemma A.4.1,

$$|\Gamma|_{\Delta} \vdash |\mathtt{e}'|_{\Delta,\Gamma} \overset{\text{exp}}{\Longrightarrow} [|\mathtt{\bar{d}}|_{\Delta,\Gamma}/\mathtt{\bar{x}}, |\mathtt{new\ C<\bar{T}>(\bar{e})}|_{\Delta,\Gamma}/\mathtt{this}]\mathtt{e_0}'.$$

Note that $|\mathtt{e}'|_{\Delta,\Gamma} = [|\mathtt{\bar{d}}|_{\Delta,\Gamma}/\mathtt{\bar{x}}, |\mathtt{new\ C<\bar{T}>(\bar{e})}|_{\Delta,\Gamma}/\mathtt{this}]|\mathtt{e_0}|_{\Delta,\Gamma'}$. By Theorems 3.4.1 and 4.5.1,

$$|\Gamma|_{\Delta} \vdash_{\mathrm{FJ}} |\mathtt{e}'|_{\Delta,\Gamma} : |\mathtt{T}'|_{\Delta}$$

for some $\mathtt{T}'$ such that $\Delta \vdash \mathtt{T}' <:_{\mathrm{FGJ}} \mathtt{T}$. By Lemma A.3.5, $|\mathtt{T}'|_{\Delta} <:_{\mathrm{FJ}} \mathtt{D}$. Thus,

$$|\Gamma|_{\Delta} \vdash |\mathtt{e}'|_{\Delta,\Gamma} \overset{\text{exp}}{\Longrightarrow} \mathtt{(D)}^{s}|\mathtt{e}'|_{\Delta,\Gamma}.$$

Finally,

$$|\Gamma|_{\Delta} \vdash |\mathtt{e}'|_{\Delta,\Gamma} \overset{\text{exp}}{\Longrightarrow} \mathtt{(D)}^{s}[|\mathtt{\bar{d}}|_{\Delta,\Gamma}/\mathtt{\bar{x}}, |\mathtt{new\ C<\bar{T}>(\bar{e})}|_{\Delta,\Gamma}/\mathtt{this}]\mathtt{e_0}'.$$

Subcase E-INVK. Similarly to the subcase above.

Case GR-CAST. Easy. □

LEMMA A.4.5. *If* $\Gamma \vdash_{\mathrm{FJ}} \mathtt{e} : \mathtt{C}$ *and* $\mathtt{e} \rightarrow_{\mathrm{FJ}} \mathtt{e}'$ *and* $\Gamma \vdash \mathtt{e} \overset{\text{exp}}{\Rightarrow} \mathtt{d}$, *then there exists some FJ expression* $\mathtt{d}'$ *such that* $\Gamma \vdash \mathtt{e}' \overset{\text{exp}}{\Rightarrow} \mathtt{d}'$ *and* $\mathtt{d} \rightarrow_{\mathrm{FJ}}{}^{*} \mathtt{d}'$. *In other words, the diagram shown in Figure* 13 *commutes.*

PROOF. By induction on the derivation of $\mathtt{e} \rightarrow_{\mathrm{FJ}} \mathtt{e}'$ with a case analysis on the last reduction rule used.

Case R-FIELD. $\mathtt{e} = \mathtt{new\ C(\bar{e}).f}_i$ $\quad \mathit{fields}_{\mathrm{FJ}}(\mathtt{C}) = \mathtt{\bar{C}\ \bar{f}}$ $\quad \mathtt{e}' = \mathtt{e}_i$

$$\begin{tikzcd}
\mathtt{e} \arrow[rr, "\text{reduce (FGJ)}"] \arrow[dddd, "\text{erase}"'] & & \mathtt{e}' \arrow[rr, "\text{reduce (FGJ)}_{*}"] \arrow[dd, dotted, "\text{erase}"'] & & \mathtt{e}'' \arrow[d, dotted, "\text{erase}"] \\
& & & (2) & |\mathtt{e}''| \arrow[d, dotted, Rightarrow] \\
& (1) & |\mathtt{e}'| \arrow[rr, dotted, "\text{reduce (FJ)}^{*}"] \arrow[dd, dotted, Rightarrow] & & \mathtt{d}' \arrow[dd, dotted, Rightarrow] \\
& & & (3) & \\
|\mathtt{e}| \arrow[rr, dotted, "\text{reduce (FJ)}"'] & & \mathtt{d} \arrow[rr, dotted, "\text{reduce (FJ)}"', "*"] & & \mathtt{d}''
\end{tikzcd}$$

Fig. 14.

The expansion $\mathtt{d}$ must have a form of $\mathtt{((D_1)^s \cdots (D_n)^s new\ C(\bar{d})).f}_i$ where $\Gamma \vdash \bar{\mathtt{e}} \overset{\text{exp}}{\Rightarrow} \bar{\mathtt{d}}$ and $\mathtt{C}\texttt{<:}_{\text{FJ}}\mathtt{D}_i$ for $1 \le i \le n$ because each $\mathtt{D}_i$ is introduced as an upcast. Thus, $\mathtt{d} \rightarrow_{\text{FJ}}{}^* \mathtt{new\ C(\bar{d}).f}_i \rightarrow_{\text{FJ}} \mathtt{d}_i$.

The other base cases are similar, and the cases for induction steps are straightforward. □

Proof of Theorem 4.5.3. By induction on the length n of reduction sequence $\mathtt{e} \rightarrow_{\text{FGJ}}{}^* \mathtt{e}'$.

Case. $n = 0$

Trivial.

Case. $\mathtt{e} \rightarrow_{\text{FGJ}} \mathtt{e}' \rightarrow_{\text{FGJ}}{}^* \mathtt{e}''$

We have the commuting diagram shown in Figure 14. Commutation (1) is proved by Lemma A.4.4, (2) by the induction hypothesis and (3) by Lemma A.4.5. □

Lemma A.4.6. *Suppose* $\Delta; \Gamma \vdash_{\text{FGJ}} \mathtt{e} : \mathtt{T}$. *If* $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\text{FJ}} \mathtt{d}$, *then* $\mathtt{e} \rightarrow_{\text{FGJ}} \mathtt{e}'$ *for some* $\mathtt{e}'$ *and* $|\Gamma|_\Delta \vdash |\mathtt{e}'|_{\Delta,\Gamma} \overset{\text{exp}}{\Rightarrow} \mathtt{d}$. *In other words, the diagram in Figure* 15 *commutes.*

Proof. By induction on the derivation of $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\text{FJ}} \mathtt{d}$ with a case analysis by the last rule used. We show only a few main cases.

Case RC-Cast. We have two subcases according to whether the cast is synthetic ($|\mathtt{e}|_{\Delta,\Gamma} = \mathtt{(C)}^s\mathtt{e}_0$) or not ($|\mathtt{e}|_{\Delta,\Gamma} = \mathtt{(C)e}_0$). The latter case follows from the induction hypothesis. We show the former case, where

$$\begin{aligned}
&|\mathtt{e}|_{\Delta,\Gamma} = \mathtt{(C)}^s\mathtt{e}_0\\
&\mathtt{e}_0 \rightarrow_{\text{FJ}} \mathtt{d}_0\\
&\mathtt{d} = \mathtt{(C)}^s\mathtt{d}_0.
\end{aligned}$$

Then $\mathtt{e}_0$ must be either a field access or a method invocation. We have another case analysis with the last reduction rule for the derivation of $\mathtt{e}_0 \rightarrow_{\text{FJ}} \mathtt{d}_0$. The cases for RC-Field, RC-Invk-Recv, and RC-Invk-Arg are omitted, since they

$$\begin{tikzcd}
\mathtt{e} \arrow[r, dotted, "\text{reduce (FGJ)}"] \arrow[dd, "\text{erase}"'] & \mathtt{e}' \arrow[d, dotted, "\text{erase}"] \\
& |\mathtt{e}'| \arrow[d, Rightarrow, dotted] \\
|\mathtt{e}| \arrow[r, "\text{reduce (FJ)}"'] & \mathtt{d}
\end{tikzcd}$$

Fig. 15.

follow from the induction hypothesis.

Subcase R-FIELD. $\mathtt{e}_0 = \mathtt{new\ D(\bar{e}).f}_i$
$\mathtt{d}_0 = \mathtt{e}_i$
$\mathit{fields}_{\mathrm{FJ}}(\mathtt{D}) = \bar{\mathtt{C}}\ \bar{\mathtt{f}}$

By inspecting the derivation of $|\mathtt{e}|_{\Delta,\Gamma}$, it must be the case that

$$\begin{aligned}
&\mathtt{e} = \mathtt{new\ D\text{<}\bar{T}\text{>}(\bar{e}').f}_i\\
&|\bar{\mathtt{e}}'|_{\Delta,\Gamma} = \bar{\mathtt{e}}\\
&\mathit{fieldsmax}(\mathtt{D}) = \bar{\mathtt{C}}\ \bar{\mathtt{f}}\\
&|\mathtt{T}|_\Delta = \mathtt{C} \neq \mathtt{C}_i.
\end{aligned}$$

By Theorems 3.4.2 and 3.4.1, we have $\mathtt{e} \rightarrow_{\mathrm{FGJ}} \mathtt{e}_i{}'$ and $\Delta;\Gamma \vdash_{\mathrm{FGJ}} \mathtt{e}_i{}'\mathtt{:S}$ and $\Delta \vdash \mathtt{S}$ <:$_{\mathrm{FGJ}}$ $\mathtt{T}$. By Theorem 4.5.1, $|\Gamma|_\Delta \vdash_{\mathrm{FJ}} |\mathtt{e}_i{}'|_{\Delta,\Gamma}\mathtt{:}|\mathtt{S}|_\Delta$. By Lemma A.3.5, $|\mathtt{S}|_\Delta$ <:$_{\mathrm{FJ}}$ $|\mathtt{T}|_\Delta$. Then, $|\Gamma|_\Delta \vdash \mathtt{e}_i \overset{\mathrm{exp}}{\Rightarrow} (|\mathtt{T}|_\Delta)\mathtt{e}_i$, finishing the case.

Subcase R-INVK. $\mathtt{e}_0 = \mathtt{new\ D(\bar{d}).m(\bar{e})}$
$\mathit{mbody}_{\mathrm{FJ}}(\mathtt{m},\mathtt{D}) = \bar{\mathtt{x}}.\mathtt{e}_m$
$\mathtt{d}_0 = [\bar{\mathtt{e}}/\bar{\mathtt{x}}, \mathtt{new\ D(\bar{d})/this}]\mathtt{e}_m$

By inspecting the derivation of $|\mathtt{e}|_{\Delta,\Gamma}$, it must be the case that

$$\begin{array}{lll}
\mathtt{e} = \mathtt{new\ D\text{<}\bar{T}\text{>}(\bar{d}').m\text{<}\bar{V}\text{>}(\bar{e}')} & |\bar{\mathtt{d}}'|_{\Delta,\Gamma} = \bar{\mathtt{d}} & |\bar{\mathtt{e}}'|_{\Delta,\Gamma} = \bar{\mathtt{e}}\\
\mathit{mtype}_{\mathrm{FGJ}}(\mathtt{m},\mathtt{D\text{<}\bar{T}\text{>}}) = \mathtt{\text{<}\bar{Y} \triangleleft \bar{P}\text{>}\bar{U}{\rightarrow}U_0} & [\bar{\mathtt{V}}/\bar{\mathtt{Y}}]\mathtt{U}_0 = \mathtt{T} & \\
\mathit{mtypemax}(\mathtt{m},\mathtt{D}) = \bar{\mathtt{C}}{\rightarrow}\mathtt{C}_0 & |\mathtt{T}|_\Delta = \mathtt{C} \neq \mathtt{C}_0. &
\end{array}$$

By Theorems 3.4.2. and 3.4.1, it must be the case that

$$\begin{aligned}
&\mathtt{e} \rightarrow_{\mathrm{FGJ}} [\bar{\mathtt{e}}'/\bar{\mathtt{x}}, \mathtt{new\ D\text{<}\bar{T}\text{>}(\bar{d}')/this}]\mathtt{e}_m{}'\\
&\mathit{mbody}_{\mathrm{FGJ}}(\mathtt{m\text{<}\bar{V}\text{>}}, \mathtt{D\text{<}\bar{T}\text{>}}) = \bar{\mathtt{x}}.\mathtt{e}_m{}'\\
&\Delta;\Gamma \vdash_{\mathrm{FGJ}} [\bar{\mathtt{e}}'/\bar{\mathtt{x}}, \mathtt{new\ D\text{<}\bar{T}\text{>}(\bar{d}')/this}]\mathtt{e}_m{}'\mathtt{:S}
\end{aligned}$$

for some $\mathtt{S}$ such that $\Delta \vdash \mathtt{S}$<:$\mathtt{T}$. By Theorem 4.5.1 and the fact that

$$|[\bar{\mathtt{e}}'/\bar{\mathtt{x}}, \mathtt{new\ D\text{<}\bar{T}\text{>}(\bar{d}')/this}]\mathtt{e}_m{}'|_{\Delta,\Gamma} = [\bar{\mathtt{e}}/\bar{\mathtt{x}}, \mathtt{new\ D(\bar{d})/this}]|\mathtt{e}_m{}'|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{D\text{<}\bar{T}\text{>}}}$$

where $\bar{\mathtt{W}}$ are the types of $\bar{\mathtt{e}}'$, we have

$$|\Gamma|_\Delta \vdash_{\mathrm{FJ}} [\bar{\mathtt{e}}/\bar{\mathtt{x}}, \mathtt{new\ D(\bar{d})/this}]|\mathtt{e}_m{}'|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{D\text{<}\bar{T}\text{>}}}\mathtt{:}|\mathtt{S}|_\Delta.$$

Since $|\mathtt{S}|_\Delta$ <:$_{\mathrm{FJ}}$ $|\mathtt{T}|_\Delta$ by Lemma A.3.5,

$$\begin{aligned}
&|\Gamma|_\Delta \vdash [\bar{\mathtt{e}}/\bar{\mathtt{x}}, \mathtt{new\ D(\bar{d})/this}]|\mathtt{e}_m{}'|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{D\text{<}\bar{T}\text{>}}}\\
&\quad \overset{\mathrm{exp}}{\Rightarrow} (|\mathtt{T}|_\Delta)^s[\bar{\mathtt{e}}/\bar{\mathtt{x}}, \mathtt{new\ D(\bar{d})/this}]|\mathtt{e}_m{}'|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{D\text{<}\bar{T}\text{>}}}.
\end{aligned}$$

$$\begin{tikzcd}
{|\mathtt{e}|} \arrow[rr, dotted, "\text{reduce (FJ)}"] \arrow[d, Rightarrow] & & \mathtt{e}' \arrow[d, dotted] \\
\mathtt{d} \arrow[r, "\text{R-Cast}"', "*"] & {} \arrow[r, "\text{reduce (FJ)}"'] & \mathtt{d}'
\end{tikzcd}$$

Fig. 16.

On the other hand, by Lemma A.4.3,

$$|\bar{\mathtt{x}} : \bar{\mathtt{W}}, \mathtt{this} : \mathtt{D}\texttt{<}\bar{\mathtt{T}}\texttt{>}|_\Delta \vdash |\mathtt{e}_m{}'|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{D}\texttt{<}\bar{\mathtt{T}}\texttt{>}} \overset{\text{exp}}{\Rightarrow} \mathtt{e}_m.$$

By Lemma A.4.1,

$$|\Gamma|_\Delta \vdash [\bar{\mathtt{e}}/\bar{\mathtt{x}}, \mathtt{new\ D}(\bar{\mathtt{d}})/\mathtt{this}]|\mathtt{e}_m{}'|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{D}\texttt{<}\bar{\mathtt{T}}\texttt{>}} \overset{\text{exp}}{\Rightarrow} [\bar{\mathtt{e}}/\bar{\mathtt{x}}, \mathtt{new\ D}(\bar{\mathtt{d}})/\mathtt{this}]\mathtt{e}_m.$$

Then,

$$\begin{aligned}|\Gamma|_\Delta \vdash (|\mathtt{T}|_\Delta)^s[\bar{\mathtt{e}}/\bar{\mathtt{x}}, \mathtt{new\ D}(\bar{\mathtt{d}})/\mathtt{this}]|\mathtt{e}_m{}'|_{\Delta,\bar{\mathtt{x}}:\bar{\mathtt{W}},\mathtt{this}:\mathtt{D}\texttt{<}\bar{\mathtt{T}}\texttt{>}}\\ \overset{\text{exp}}{\Rightarrow} (|\mathtt{T}|_\Delta)^s[\bar{\mathtt{e}}/\bar{\mathtt{x}}, \mathtt{new\ D}(\bar{\mathtt{d}})/\mathtt{this}]\mathtt{e}_m.\end{aligned}$$

Finally, we have, by the fact that $\mathtt{C} = |\mathtt{T}|_\Delta$ and transitivity of the expansion relation,

$$|\Gamma|_\Delta \vdash |[\bar{\mathtt{e}}'/\bar{\mathtt{x}}, \mathtt{new\ D}\texttt{<}\bar{\mathtt{T}}\texttt{>}(\bar{\mathtt{d}}')/\mathtt{this}]\mathtt{e}_m{}'|_{\Delta,\Gamma} \overset{\text{exp}}{\Rightarrow} (\mathtt{C})[\bar{\mathtt{e}}/\bar{\mathtt{x}}, \mathtt{new\ D}(\bar{\mathtt{d}})/\mathtt{this}]\mathtt{e}_m{}'.$$

Case R-Field. Similar to the subcase for R-Field in the case for RC-Cast above.

Case R-Invk. Similar to the subcase for R-Invk in the case for RC-Cast above. The case for R-Cast and the other cases for induction steps are straightforward. □

Lemma A.4.7. *Suppose* $\Delta; \Gamma \vdash_{\text{FGJ}} \mathtt{e} : \mathtt{T}$ *and* $|\Gamma|_\Delta \vdash |\mathtt{e}|_{\Delta,\Gamma} \overset{\text{exp}}{\Rightarrow} \mathtt{d}$. *If* $\mathtt{d}$ *reduces to* $\mathtt{d}'$ *with zero or more steps by removing synthetic casts, followed by one step by other kinds of reduction, then* $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\text{FJ}} \mathtt{e}'$ *and* $|\Gamma|_\Delta \vdash \mathtt{e}' \overset{\text{exp}}{\Rightarrow} \mathtt{d}'$. *In other words, the diagram in Figure* 16 *commutes.*

Proof. By induction on the derivation of the last reduction step with a case analysis by the last rule used.

Case R-Field. $\mathtt{d} \rightarrow_{\text{FJ}}{}^* \mathtt{new\ C}(\bar{\mathtt{e}}).\mathtt{f}_i$ $\mathit{fields}_{\text{FJ}}(\mathtt{C}) = \bar{\mathtt{C}}\ \bar{\mathtt{f}}$ $\mathtt{d}' = \mathtt{e}_i$

The expression $\mathtt{d}$ must be of the form $((\mathtt{D}_1)^s \ldots (\mathtt{D}_n)^s\mathtt{new\ C}(\bar{\mathtt{e}}')).\mathtt{f}_i$ where $\mathtt{C} \texttt{<:} \mathtt{D}_i$ for any i and each $\mathtt{e}_i{}'$ reduces to $\mathtt{e}_i$ by removing upcasts (in several steps). In other words, $|\Gamma|_\Delta \vdash \bar{\mathtt{e}}' \overset{\text{exp}}{\Rightarrow} \bar{\mathtt{e}}$. Moreover, since $|\Gamma|_\Delta \vdash |\mathtt{e}|_{\Delta,\Gamma} \overset{\text{exp}}{\Rightarrow} \mathtt{d}$, the expression $|\mathtt{e}|_{\Delta,\Gamma}$ must be of either the form $\mathtt{new\ C}(\bar{\mathtt{e}}'').\mathtt{f}_i$ or $(\mathtt{D})^s\mathtt{new\ C}(\bar{\mathtt{e}}'').\mathtt{f}_i$, where $|\Gamma|_\Delta \vdash \bar{\mathtt{e}}'' \overset{\text{exp}}{\Rightarrow} \bar{\mathtt{e}}'$. Therefore, $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\text{FJ}} \mathtt{e}_i{}''$ or $|\mathtt{e}|_{\Delta,\Gamma} \rightarrow_{\text{FJ}} (\mathtt{D})^s\mathtt{e}_i{}''$. It is easy to see

$$|\Gamma|_\Delta \vdash (\mathtt{D})^s\mathtt{e}_i{}'' \overset{\text{exp}}{\Rightarrow} \mathtt{e}_i$$

and

$$|\Gamma|_\Delta \vdash \mathtt{e}_i{}'' \overset{\text{exp}}{\Rightarrow} \mathtt{e}_i.$$

Other base cases are similar; induction steps are straightforward. □

PROOF OF THEOREM 4.5.4. Follows from Lemmas A.4.6 and A.4.7. □

ACKNOWLEDGMENTS

We thank Robert Harper and the anonymous referees of OOPSLA'99 and TOPLAS for their valuable comments and suggestions.

REFERENCES

ABADI, M. AND CARDELLI, L. 1996. *A Theory of Objects*. Springer Verlag, New York, NY.

AGESEN, O., FREUND, S. N., AND MITCHELL, J. C. 1997. Adding type parameterization to the Java language. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (*OOPSLA'97*). *SIGPLAN Not. 32*, 10. ACM Press, Atlanta, GA, 49–65.

ANCONA, D. AND ZUCCA, E. 2001. True modules for Java-like languages. In *Proceedings of the 15th European Conference on Object-Oriented Programming* (*ECOOP2001*), J. L. Knudsen, Ed. Lecture Notes in Computer Science. Springer-Verlag, Budapest, Hungary.

BARENDREGT, H. P. 1984. *The Lambda Calculus*, revised ed. North Holland, Amsterdam, The Netherlands.

BONO, V. AND FISHER, K. 1998. An imperative first-order calculus with object extension. In *Proceedings of the 12th European Conference on Object-Oriented Programming* (*ECOOP'98*), E. Jul, Ed. LNCS 1445. Springer Verlag, 462–497.

BONO, V., PATEL, A. J., AND SHMATIKOV, V. 1999a. A core calculus of classes and mixins. In *Proceedings of the 13th European Conference on Object-Oriented Programming* (*ECOOP'99*). LNCS 1628. Springer Verlag, 43–66.

BONO, V., PATEL, A. J., SHMATIKOV, V., AND MITCHELL, J. C. 1999b. A core calculus of classes and objects. In *Proceedings of the 15th Conference on the Mathematical Foundations of Programming Semantics* (*MFPS XV*). Elsevier. Available through `http://www.elsevier.nl/locate/entcs/volume20.html`.

BRACHA, G., ODERSKY, M., STOUTAMIRE, D., AND WADLER, P. 1998. Making the future safe for the past: Adding genericity to the Java programming language. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (*OOPSLA'98*), C. Chambers, Ed. ACM Press, New York, NY, 183–200.

BRUCE, K. B. 1994. A paradigmatic object-oriented programming language: Design, static typing and semantics. *J. Funct. Program. 4*, 2 (April), 127–206.

CARDELLI, L., MARTINI, S., MITCHELL, J. C., AND SCEDROV, A. 1994. An extension of system F with subtyping. *Inf. Comput. 109*, 1–2, 4–56.

CARTWRIGHT, R. AND STEELE JR., G. L. 1998. Compatible genericity with run-time types for the Java programming language. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (*OOPSLA'98*), C. Chambers, Ed. ACM Press, New York, NY, 201–215.

DROSSOPOULOU, S., EISENBACH, S., AND KHURSHID, S. 1999. Is the Java type system sound? *Theory Pract. Object Syst. 7*, 1, 3–24.

DUGGAN, D. 1999. Modular type-based reverse engineering of parameterized types in Java code. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (*OOPSLA'99*), L. M. Northrop, Ed. ACM Press, New York, NY, 97–113.

FELLEISEN, M. AND FRIEDMAN, D. P. 1998. *A Little Java, A Few Patterns*. MIT Press, Cambridge, MA.

FISHER, K. AND MITCHELL, J. C. 1998. On the relationship between classes, objects, and data abstraction. *Theory Pract. Object Syst. 4*, 1, 3–25.

FLATT, M., KRISHNAMURTHI, S., AND FELLEISEN, M. 1998a . Classes and mixins. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, New York, NY, 171–183.

Flatt, M., Krishnamurthi, S., and Felleisen, M. 1998b. A programmer's reduction semantics for classes and mixins. Tech. Rep. TR97-293, Computer Science Dept., Rice University. Feb. Corrected version in June, 1999.

Igarashi, A. and Pierce, B. C. 2000. On inner classes. In *Proceedings of the 14th European Conference on Object-Oriented Programming* (*ECOOP2000*), E. Bertino, Ed. LNCS 1850. Springer Verlag, 129–153. Extended version to appear in *Inf. Comput.*

Igarashi, A., Pierce, B. C., and Wadler, P. 2001. A recipe for raw types. In *Informal Proceedings of the 8th International Workshop on Foundations of Object-Oriented Languages* (*FOOL8*). London, UK. Available through `http://www.cs.williams.edu/~kim/FOOL/FOOL8.html`.

League, C., Trifonov, V., and Shao, Z. 2001. Type-preserving compilation of Featherweight Java. In *Informal Proceedings of the 8th International Workshop on Foundations of Object-Oriented Languages* (*FOOL8*). London, UK. Available through `http://www.cs.williams.edu/~kim/FOOL/FOOL8.html`.

Myers, A. C., Bank, J. A., and Liskov, B. 1997. Parameterized types for Java. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, New York, NY, 132–145.

Nipkow, T. and von Oheimb, D. 1998. Java$_{light}$ is type-safe — definitely. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, New York, NY, 161–170.

Odersky, M. and Wadler, P. 1997. Pizza into Java: Translating theory into practice. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, New York, NY, 146–159.

Pierce, B. C. 2002. *Types and Programming Languages*. MIT Press, Cambridge, MA.

Pierce, B. C. and Turner, D. N. 1994. Simple type-theoretic foundations for object-oriented programming. *J. Funct. Program. 4,* 2 (April), 207–247.

Schultz, U. 2001. Partial evaluation for class-based object-oriented languages. In *Proceedings of the 2nd Symposium on Programs as Data Objects* (*PADO II*), O. Danvy and A. Filinski, Eds. LNCS 2053. Springer Verlag, 173–197.

Studer, T. 2000. Constructive foundations for Featherweight Java. Available through `http://iamwww.unibe.ch/~tstuder/`.

Syme, D. 1997. Proving Java type soundness. Tech. Rep. 427, Computer Lab. Univ. of Cambridge. June.

Viroli, M. and Natali, A. 2000. Parametric polymorphism in Java: an approach to translation based on reflective features. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (*OOPSLA'00*), D. Lea, Ed. ACM Press, New York, NY, 146–165.

Wand, M. 1989. Type inference for objects with instance variables and inheritance. Tech. Rep. NU-CCS-89-2, College of Computer Science, Northeastern Univ. Feb.

Wright, A. K. and Felleisen, M. 1994. A syntactic approach to type soundness. *Inf. Comput. 115,* 1 (Nov.), 38–94.

Received July 2000; accepted December 2000