

The Soundness of Component NextGen

James Sasitorn Robert Cartwright
camus@rice.edu cork@rice.edu
Rice University
6100 S. Main St.
Houston TX 77005

July 21, 2007

Abstract

This technical report presents a type soundness proof for Core CGEN, a small, formal language designed for studying the addition of a component system based on genericity to a nominally typed object-oriented language supporting first-class generics. Core CGEN captures the most intricate aspects of the Component NEXTGEN programming language, an extension of the NEXTGEN architecture that provides efficient support for first-class generics in Java, and maintains full compatibility with existing Java Virtual Machines. We begin by reviewing the semantics of Core CGEN, and then proceed by establishing several key lemmas. Finally, we conclude by establishing preservation and progress theorems.

1 Introduction

The Component NEXTGEN programming language is an extension to Generic Java that adds support for components, while maintaining full compatibility with existing Java Virtual Machines (JVMs). CGEN is an extension of the NEXTGEN language, a generalization of Java 5.0 that efficiently supports first-class generic types [4]. We have established that the CGEN language design constitutes a feasible extension of Generic Java, by describing how to implement it efficiently on top of the JVM in [6]. However, the new language features of CGEN introduce many subtle, technical issues. In particular, inheritance across component boundaries can produce unexpected results. The name of a method introduced in a class may collide with the name of a *public* or *protected* method in an imported superclass. To address this “accidental method capture” problem, CGEN uses a more intricate semantics for method lookup, in comparison to conventional Java. Furthermore, by leveraging the nominal type system used for generic types, CGEN can provide seamless support for mutual recursive modules. But support for mutually recursive modules allows for cyclic class hierarchies to be generated during module linking. Because of these subtleties, proofs of type soundness from FJ, FGJ, and CMG can not be directly applied to CCG. In this technical report, we argue that the CGEN design is sound by establishing a type soundness result for Core CGEN, a small formal model of the CGEN language that captures the most subtle properties of the full language. Our presentation of this proof assumes knowledge of the CGEN language design, as presented in [6, 5]. The presentation of Core CGEN semantics in the preceding sections is a review of the semantics presented in [5].

2 Core CGEN

In order to identify and resolve the subtle technical issues in the CGEN type system, we have distilled the component framework into a small, core language called Core CGEN, CCG for short. Indeed, many issues described in [5] were uncovered during a formal analysis of Component NEXTGEN.

The design of Core CGEN is based on Featherweight GJ (FGJ) [3] and incorporates ideas from Core MixGen [1]. In the remainder of this paper, we will refer to these two languages as FGJ, and CMG, respectively. CCG excludes signatures for the sake of simplicity; degenerate modules are used in place of signatures. This simplification follows the precedent established in Featherweight Java, FGJ, and CMG which exclude interfaces and rely on degenerate classes in their stead.

CCG augments FGJ with the essential infrastructure to support CGEN-style components:

- **module** definitions. These provide the crucial framework to bundle classes as components.
- **bind** definitions. These provide the ability to link (instantiate) modules with their dependencies. For simplicity, CCG programs define a single set of bind declarations.
- Multiple constructors in class definitions. In FGJ each class defines a default constructor that takes an initial value for each field as an argument. In CCG, we relax this restriction and permit multiple constructors with arbitrary signatures. This allows (i) classes to satisfy the constraints required by multiple bounding signatures, and (ii) different module implementations to provide different collections of fields.

The following sections outline the syntax, type system, small-step semantics, and proof of type soundness for CCG; the details of the proof are presented in Sections 2.9. The semantics and proof are similar to those for MixGen in [2], except for the following:

- All class types in CCG are prefixed by their enclosing module instantiation. This convention is analogous to using fully-qualified class names including a package prefix in Java.
- CCG must check that class hierarchies form a DAG when modules are linked together with respect to the declared bind declarations.
- The small-step semantics for CCG carries a runtime “bound” environment, mapping type variables to their bounds, to support bind declarations.

2.1 Syntax

The abstract syntax of CCG, shown in Table 1, consists of module declarations (**MD**), class declarations (**CL**), constructors declarations (**K**), method declarations (**M**), expressions (**e**), types (**T**), and bind declarations (**BD**). For the sake of brevity, **extends** is abbreviated by \triangleleft . Throughout all formal rules of the language, the following meta-variables are used over the following domains:

- **d**, **e** range over expressions.
- **K** ranges over constructors.
- **m**, **M** range over methods.
- **N**, **O**, **P** range over fully-qualified class types
- **N**, **O**, **P** range over module types

Syntax:	
MD	::= module $\mathbb{D} \langle \bar{X} \triangleleft \bar{N} \rangle \triangleleft N \{ \bar{CL} \}$
CL	::= class $C \langle \bar{X} \triangleleft \bar{N} \rangle \triangleleft N \{ \bar{T} \bar{f}; \bar{K} \bar{M} \}$
K	::= $C(\bar{T} \bar{x}) \{ \text{super}(\bar{e}); \text{this}.\bar{f} = \bar{e}; \}$
M	::= $\langle \bar{X} \triangleleft \bar{N} \rangle T m(\bar{T} \bar{x}) \{ \text{return } e; \}$
e	::= $\begin{array}{l} x \\ \\ e.f \\ \\ e.m \langle \bar{T} \rangle (\bar{e}) \\ \\ \text{new } N(\bar{e}) \\ \\ (N)e \end{array}$
T	::= $X \mid N$
\bar{T}	::= $\bar{X} \mid \bar{N}$
N	::= $T.C \langle \bar{T} \rangle$
\bar{N}	::= $\mathbb{D} \langle \bar{T} \rangle$
BD	::= bind $N \bar{X} = N;$

Table 1: CCG Syntax

- X, Y, Z range over naked class type variables.
- $\bar{X}, \bar{Y}, \bar{Z}$ range over naked module type variables.
- R, S, T, U, V range over class types.
- $\mathbb{R}, \mathbb{S}, \mathbb{T}, \mathbb{U}, \mathbb{V}$ range over module types.
- x ranges over method parameter names.
- f ranges over field names.
- C, D range over class names.
- \mathbb{C}, \mathbb{D} range over module names

Following the notation of FGJ, a meta-variable with a horizontal bar above it represents a (possibly empty) sequence of elements in the domain of that variable, and may include an arbitrary separator character. For example, \bar{T} denotes a sequence of types T_0, \dots, T_n . As in CMG, we abuse this notation in different contexts. For example, $\bar{T} \bar{f};$ denotes $T_0 f_0; \dots, T_n f_n;$. Similarly, the expression $\bar{X} \triangleleft \bar{N}$ represents $X_0 \triangleleft N_0, \dots, X_n \triangleleft N_n$.

In CCG, sequences of classes, type variables, method names, and field names are required to contain no duplicates. The set of type variables in each module includes an implicit variable `thisMod` which cannot appear as a type parameter anywhere in the module. The set of variables in each class includes an implicit variable `this` which cannot appear as a class name, field, or method parameter anywhere in the class.

Type variable bounds may reference other type parameters declared in the same scope; in other words they may be mutually recursive. Every module definition declares a super module using \triangleleft . Every class definition declares a super class using \triangleleft .

CCG requires explicit polymorphism on all parametric method invocations.

2.2 Valid Programs

A CCG program consists of a fixed module table, a fixed bind table and an expression, denoted (MT, BT, e) . A module table MT is a mapping from module names \mathbb{D} to module declarations \mathbb{MD} . A bind table BT is a mapping from type variables \mathbb{X} to bind declarations \mathbb{BD} .

A valid module table MT must satisfy the following constraints: (i) for every \mathbb{D} in MT , $MT(\mathbb{D}) = \text{module } \mathbb{D} \dots$, (ii), $\text{Mod} \notin \text{dom}(MT)$, (iii) every module appearing in MT is in $\text{dom}(MT)$, and (iv) the subtyping relation $<$: induced by MT is antisymmetric and forms a tree rooted at Mod . The root module Mod is modeled without a corresponding module definition in the module table and contains no classes.

A valid bind table BT must satisfy the following constraints: (i) for every \mathbb{X} in BT , $BT(\mathbb{X}) = \text{bind } \mathbb{N}_0 \mathbb{X} = \mathbb{N}$; and (ii) every type variable \mathbb{X} appearing in BT is in $\text{dom}(BT)$. Given the set BT of binds $\text{bind } \overline{\mathbb{N}}_0 \overline{\mathbb{X}} = \overline{\mathbb{N}}$, the initial bound environment, mapping module type variables to their upper bounds is defined as $\Delta_{BT} = \overline{\mathbb{X}} \triangleleft \overline{\mathbb{N}}$. The bound environment Δ is discussed below in Section 2.4.

Program execution consists of evaluating e in the context of MT and the initial bound environment Δ_{BT} for BT .

2.3 Valid Module Binds

To determine if a set of binds BT can be safely linked to produce an acyclic set of classes¹, an implicit class table CT , defining a mapping from fully-qualified class names to definitions, is generated by evaluating MT in the presence of BT . For each bind declaration $\text{bind } \mathbb{N}_0 \mathbb{X} = \mathbb{D} \langle \overline{\mathbb{T}} \rangle$; in BT where $MT(\mathbb{D}) = \text{module } \mathbb{D} \langle \overline{\mathbb{Y}} \triangleleft \overline{\mathbb{N}} \rangle \triangleleft \mathbb{N} \{ \overline{\mathbb{CL}} \}$ and each class $\text{class } \mathbb{C} \langle \overline{\mathbb{X}} \triangleleft \overline{\mathbb{N}} \rangle \triangleleft \mathbb{N} \{ \dots \} \in \overline{\mathbb{CL}}$:

$$CT(\mathbb{D} \langle \overline{\mathbb{T}} \rangle . \mathbb{C}) = [\overline{\mathbb{Y}} \mapsto \text{bound}_{\Delta_{BT}}(\overline{\mathbb{T}})] \\ \text{class } \mathbb{D} \langle \overline{\mathbb{Y}} \rangle . \mathbb{C} \langle \overline{\mathbb{X}} \triangleleft \overline{\mathbb{N}} \rangle \triangleleft \mathbb{N} \{ \dots \}.$$

The substitution above replaces module type parameters with their *bind*-ed instantiations so that the parent type of a class in an instantiated module can be looked up in CT . The domain of CT is finite; It simply consists of the set of classes defined in the module instantiations in the right hand sides of **bind** declarations.²

A valid class table must satisfy the following constraints: (i) every class name appearing in CT is in $\text{dom}(CT)$ and (ii) the set of class definitions must form a tree rooted at **Object**.³

The class **Object** is modeled as a top-level construct, located outside any module. **Object** contains no fields or methods and it acts as if it contains a special, zero-ary constructor.

2.4 Type Checking

The typing rules for expressions, method declarations, constructor declarations, class declarations, and module declarations are shown in Table 2. The typing rules in CCG includes two environments:

- A bound environment Δ mapping type variables to their upper bounds. Syntactically, this is written as $\overline{\mathbb{X}} \triangleleft \overline{\mathbb{N}} \cup \overline{\mathbb{X}} \triangleleft \overline{\mathbb{N}}$. The bound of a type variable is always a non-variable. The bound of a non-variable module type \mathbb{N} is \mathbb{N} , and a non-variable class type $\mathbb{V} . \mathbb{C} \langle \overline{\mathbb{T}} \rangle$ is the class type $\mathbb{C} \langle \overline{\mathbb{T}} \rangle$ prefixed by the bound of the enclosing module \mathbb{V} .

¹Since mixin classes in MIXGEN syntactically encoded their parent instantiations, formalizing properties on acyclic type hierarchies is relatively easy. CCG, just like FGJ, does not use a syntactic encoding.

²Modules *do not* contain any module type applications because they can only reference imported modules which are fully-qualified.

³The special class object is not in CT , because **Object** is a keyword.

$\Delta \vdash T <: T$ [SCReflex]	$\frac{\Delta \vdash S <: T \quad \Delta \vdash T <: U}{\Delta \vdash S <: U}$ [SCTrans]
$\Delta \vdash T <: T$ [SMReflex]	$\frac{\Delta \vdash S <: T \quad \Delta \vdash T <: U}{\Delta \vdash S <: U}$ [SMTrans]
$\Delta \vdash X <: \Delta(X)$ [SCBound]	$\Delta \vdash X <: \Delta(X)$ [SMBound]
$\begin{array}{l} bound_{\Delta}(T) = \mathbb{D} < \bar{T} > \\ MT(\mathbb{D}) = \mathbf{module} \mathbb{D} < \bar{Y} < \bar{N} > < \bar{N} \{ \bar{C} \} \\ \mathbf{class} C < \bar{X} < \bar{N} > < \bar{N} \{ \dots \} \in \bar{C} \} \\ \hline \Delta \vdash T.C < \bar{S} > <: [\bar{Y} \mapsto \bar{T}] [\bar{X} \mapsto \bar{S}] \bar{N} \\ \hline \frac{MT(\mathbb{D}) = \mathbf{module} \mathbb{D} < \bar{Y} < \bar{N} > < \bar{N} \{ \dots \} }{\Delta \vdash \mathbb{D} < \bar{T} > <: [\bar{Y} \mapsto \bar{T}] \bar{N}} \end{array}$	
<hr/> $\begin{array}{l} bound_{\Delta}(X) = \Delta(X) \\ bound_{\Delta}(V.C < \bar{T} >) = bound_{\Delta}(V).C < \bar{T} > \\ bound_{\Delta}(N) = N \end{array}$	

Table 2: Subtyping and Type Bounds

- A type environment Γ mapping program variables to their static types. Syntactically, these mappings have the form $\bar{x} : \bar{T}$

CCG contains two disjoint sets of types: module types and class types. Class types are always qualified with their enclosing module instantiation. Class type variables are bound by class types and module type variables are bound by module types. Figure 2 shows the rules for the subtyping relation $<: \cdot$. Subtyping in CCG is reflexive and transitive. Classes and modules are subtypes of the instantiations of their respective parent types.

2.5 Well-formed Types and Declarations

The rules for well-formed constructs appear in Table 3. Class and module instantiations are well-formed in the environment Δ if all instantiations of type parameters are subtypes of their bounds in Δ . Type variables are well-formed if they are present in Δ .

A method m is well-formed with respect to its enclosing module \mathbb{D} and class C if its constituent types are well-formed; the type of the body in Γ is a subtype of the declared return type; and m is a *valid override* of any method of the same name in the static type of parent class for C . An overriding method definition is valid if it preserves the signature in the parent. This notion is formalized in section 2.6.

CCG allows multiple constructors in a class. As in FGJ, there is no null value in the language, so all constructors are required to assign values to all fields. In order to avoid pathologies such as the assignment of a field to the (yet to be initialized) value of another field, all expressions in a constructor are typed in an environment binding only the constructor parameters (not the enclosing class fields, **this**, or **thisMod**).

A class definition CL is well-formed in the context of the enclosing module if the constituent elements are

well-formed; none of the fields known statically to occur in ancestors are shadowed⁴.; and every constructor has a distinct signature.

Module definitions are well-formed if their constituent elements are well-formed; each member class has a distinct name; and the module provides *valid class overrides* for all classes defined in its super module. This notion is formalized in section 2.6.

Bind declarations are well-formed if the instantiated types are well-formed and subtypes of their formal types with respect to Δ .

A program is well-formed if all module definitions are well-formed, the induced module table is well-formed, the set of binds is well-formed, and the trailing expression can be typed with empty type environment (Γ) and bound environment Δ_{BT} .

2.6 Class and Module Auxiliary Functions

The auxiliary functions defining field name and value lookup, method typing, method lookup, valid method overrides, constructor inclusion, and method inclusion appear in Table 5. The auxiliary functions defining valid class overrides appear in Table 6. These functions are passed a bound environment Δ to determine the module bounds of type parameters.

The mapping *fields* returns only the fields directly defined in a class definition. The mapping *fieldVals* is used to retrieve the field values for a given object. A static type is passed to *fieldVals* to disambiguate field names in the presence of accidental shadowing.

Method types are resolved by searching upward the class inheritance chain (which may cross modules via imports) starting from the provided static receiver type. The type of a method includes the enclosing module and class instantiation in which the method occurs, as well as the parameter types and return types. The included module and class names are used to annotate receiver expressions in the typing rules for method invocation. As explained later in Section 2.7, the annotated type of the receiver of an application of method m is reduced to a more specific type when the more specific type includes m (with a compatible method signature) in the static type of its parent. Once the annotated type of a receiver is reduced to the most specific type possible, lookup of m starts at the reduced annotated type.

For a given module \mathbb{D} , a class definition $\mathbb{C}\mathbb{L}$ defines a *valid class override* if $\mathbb{C}\mathbb{L}$ provides a super set of the fields, constructors, and methods provided by the class of the same name, if it exists, in the static type of the super module for \mathbb{D} .

2.7 Expression Typing

The rules for expression typing are given in Table 4. Naked type variables may occur in casts, but are prohibited in **new** operations.

The expression typing rules annotate the receiver expression for field lookups and method invocations with a static type. In the case of a field lookup, this static type is used to disambiguate the field reference in the presence of accidental shadowing. Although classes are statically prevented from shadowing the known fields of their ancestors, a mixin instantiation may accidentally shadow a field contained in its parent. In the case of method invocations, the receiver is annotated with a static type to allow for a “downward” search of a method definition at run-time, as explained in section 2.6.

Notice that the receiver expression of a method invocation is annotated not with its precise static type, but instead with the closest supertype of the static type in which the called method is explicitly defined. The

⁴This restriction is not necessary in principle, but it significantly simplifies the proof of type safety.

$\Delta \vdash \text{Object ok}_{[\text{WFOobject}]} \quad \Delta \vdash \text{Mod ok}_{[\text{WFMod}]}$ $\text{bound}_\Delta(\mathbb{T}) = \mathbb{D}\langle \bar{\mathbb{T}} \rangle \quad \Delta \vdash \mathbb{D}\langle \bar{\mathbb{T}} \rangle \text{ ok}$ $MT(\mathbb{D}) = \text{module } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle \triangleleft \mathbb{N} \{ \bar{\mathbb{C}}\mathbb{L} \}$ $\text{class } \mathbb{C}\langle \bar{\mathbb{X}} \triangleleft \bar{\mathbb{N}} \rangle \triangleleft \mathbb{N} \{ \dots \} \in \bar{\mathbb{C}}\mathbb{L}$ $\frac{\Delta \vdash \bar{\mathbb{S}} \text{ ok} \quad \Delta \vdash \bar{\mathbb{S}} <: [\bar{\mathbb{Y}} \mapsto \bar{\mathbb{T}}][\bar{\mathbb{X}} \mapsto \bar{\mathbb{S}}]\bar{\mathbb{N}}}{\Delta \vdash \mathbb{T}. \mathbb{C}\langle \bar{\mathbb{S}} \rangle \text{ ok}}_{[\text{WFClass}]}$	$\frac{\frac{\bar{\mathbb{X}} \in \text{dom}(\Delta)}{\Delta \vdash \bar{\mathbb{X}} \text{ ok}}_{[\text{WFVar}]} \quad \frac{\bar{\mathbb{X}} \in \text{dom}(\Delta)}{\Delta \vdash \bar{\mathbb{X}} \text{ ok}}_{[\text{WFModule}]}}{\Delta \vdash \mathbb{D}\langle \bar{\mathbb{T}} \rangle \text{ ok}}$ $MT(\mathbb{D}) = \text{module } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle \triangleleft \mathbb{N} \{ \bar{\mathbb{C}}\mathbb{L} \}$ $\Delta \vdash \bar{\mathbb{T}} \text{ ok} \quad \Delta \vdash \bar{\mathbb{T}} <: [\bar{\mathbb{Y}} \mapsto \bar{\mathbb{T}}]\bar{\mathbb{N}}$
$MT(\mathbb{D}) = \text{module } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle \triangleleft \mathbb{N} \{ \bar{\mathbb{C}}\mathbb{L} \} \quad \text{class } \mathbb{C}\langle \bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}} \rangle \triangleleft \mathbb{N} \{ \bar{\mathbb{T}} \bar{\mathbb{f}}; \bar{\mathbb{K}} \bar{\mathbb{M}} \} \in \bar{\mathbb{C}}\mathbb{L}$ $\Delta = \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} + \text{thisMod} \triangleleft \mathbb{D}\langle \bar{\mathbb{Y}} \rangle + \bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}} \quad \mathbb{N} = \mathbb{V}. \mathbb{D}\langle \bar{\mathbb{S}} \rangle \quad \text{bound}_\Delta(\mathbb{V}) = \mathbb{C}\langle \bar{\mathbb{Z}} \rangle$ $\bar{\mathbb{x}} \cap \text{this} = \emptyset \quad \Delta \vdash \text{override}(\mathbb{C}\langle \bar{\mathbb{Z}} \rangle. \mathbb{D}\langle \bar{\mathbb{S}} \rangle, \langle \bar{\mathbb{X}}' \triangleleft \bar{\mathbb{R}}' \rangle \mathbb{V}' \text{ m}(\bar{\mathbb{T}}' \bar{\mathbb{x}}))$ $\Delta_1 = \Delta + \bar{\mathbb{X}}' \triangleleft \bar{\mathbb{R}}' \quad \Gamma = \bar{\mathbb{x}} : \bar{\mathbb{T}}' + \text{this} : \mathbb{D}\langle \bar{\mathbb{Y}} \rangle. \mathbb{C}\langle \bar{\mathbb{X}} \rangle$ $\frac{\Delta_1 \vdash \bar{\mathbb{R}}' \text{ ok} \quad \Delta_1 \vdash \bar{\mathbb{R}}' <: \text{Object} \quad \Delta_1 \vdash \mathbb{V}' \text{ ok} \quad \Delta_1 \vdash \bar{\mathbb{T}}' \text{ ok} \quad \Delta_1; \Gamma \vdash \mathbf{e} \in \mathbb{U} \quad \Delta_1 \vdash \mathbb{U} <: \mathbb{V}'}{\langle \bar{\mathbb{X}}' \triangleleft \bar{\mathbb{R}}' \rangle \mathbb{V}' \text{ m}(\bar{\mathbb{T}}' \bar{\mathbb{x}}) \{ \text{return } \mathbf{e}; \} \text{ ok in } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle. \mathbb{C}\langle \bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}} \rangle}_{[\text{TMethod}]}$	
$MT(\mathbb{D}) = \text{module } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle \triangleleft \mathbb{N} \{ \bar{\mathbb{C}}\mathbb{L} \} \quad \text{class } \mathbb{C}\langle \bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}} \rangle \triangleleft \mathbb{N} \{ \bar{\mathbb{T}} \bar{\mathbb{f}}; \bar{\mathbb{K}} \bar{\mathbb{M}} \} \in \bar{\mathbb{C}}\mathbb{L}$ $\mathbb{N} = \mathbb{V}. \mathbb{D}\langle \bar{\mathbb{S}} \rangle \quad \text{bound}_\Delta(\mathbb{V}) = \mathbb{C}\langle \bar{\mathbb{Z}} \rangle$ $\Delta = \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} + \text{thisMod} \triangleleft \mathbb{D}\langle \bar{\mathbb{Y}} \rangle + \bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}} \quad \Gamma = \bar{\mathbb{x}} : \bar{\mathbb{V}} \quad \bar{\mathbb{x}} \cap \text{this} = \emptyset$ $\frac{\Delta \vdash \bar{\mathbb{V}} \text{ ok} \quad \Delta; \Gamma \vdash \bar{\mathbf{e}}' \in \bar{\mathbb{U}} \quad \vdash \mathbb{C}\langle \bar{\mathbb{Z}} \rangle. \mathbb{D}\langle \bar{\mathbb{S}} \rangle \text{ includes init}(\bar{\mathbb{U}}) \quad \Delta; \Gamma \vdash \bar{\mathbf{e}} \in \bar{\mathbb{U}} \quad \Delta \vdash \bar{\mathbb{U}} <: \bar{\mathbb{T}}}{\mathbb{C}(\bar{\mathbb{V}} \bar{\mathbb{x}}) \{ \text{super}(\bar{\mathbf{e}}'); \text{this}. \bar{\mathbf{f}} = \bar{\mathbf{e}}; \} \text{ ok in } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle. \mathbb{C}\langle \bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}} \rangle}_{[\text{TConstructor}]}$	
$MT(\mathbb{D}) = \text{module } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle \triangleleft \mathbb{N} \{ \bar{\mathbb{C}}\mathbb{L} \}$ $\bar{\mathbb{K}} \text{ ok in } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle. \mathbb{C}\langle \bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}} \rangle \quad \bar{\mathbb{M}} \text{ ok in } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle. \mathbb{C}\langle \bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}} \rangle$ $\Delta = \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} + \text{thisMod} \triangleleft \mathbb{D}\langle \bar{\mathbb{Y}} \rangle + \bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}} \quad \Delta \vdash \bar{\mathbb{R}} \text{ ok} \quad \Delta \vdash \bar{\mathbb{R}} <: \text{Object} \quad \Delta \vdash \mathbb{N} \text{ ok}$ $\Delta \vdash \bar{\mathbb{T}} \text{ ok} \quad \bar{\mathbb{X}} \cap \text{thisMod} = \emptyset \quad \bar{\mathbf{f}} \cap \text{this} = \emptyset$ $\Delta \vdash \mathbb{D}\langle \bar{\mathbb{Y}} \rangle. \mathbb{C}\langle \bar{\mathbb{X}} \rangle <: \mathbb{V} \text{ and } \Delta \vdash \text{fields}(\mathbb{V}) = \bar{\mathbb{T}}' \bar{\mathbf{f}}' \text{ implies } \mathbf{f} \cap \mathbf{f}' = \emptyset$ $K_i = \mathbb{C}(\bar{\mathbb{T}} \bar{\mathbf{x}}) \{ \dots \} \text{ and } K_j = \mathbb{C}(\bar{\mathbb{T}} \bar{\mathbf{x}}') \{ \dots \} \text{ implies } i = j$ $\frac{}{\text{class } \mathbb{C}\langle \bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}} \rangle \triangleleft \mathbb{N} \{ \bar{\mathbb{T}} \bar{\mathbf{f}}; \bar{\mathbb{K}} \bar{\mathbb{M}} \} \text{ ok in } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle}_{[\text{TClass}]}$	
$\bar{\mathbb{C}}\mathbb{L} \text{ ok in } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle \quad \bar{\mathbb{N}} <: \text{Mod} \quad \bar{\mathbb{Y}} \cap \text{thisMod} = \emptyset$ $\Delta = \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \quad \Delta \vdash \bar{\mathbb{N}} \text{ ok} \quad \Delta \vdash \mathbb{N} \text{ ok} \quad \Delta \vdash \text{classOverride}(\mathbb{N}, \bar{\mathbb{C}}\mathbb{L})$ $\mathbb{C}\mathbb{L}_i = \text{class } \mathbb{C}\langle \bar{\mathbb{X}} \triangleleft \bar{\mathbb{T}} \rangle \triangleleft \mathbb{N} \{ \dots \} \text{ and } \mathbb{C}\mathbb{L}_j = \text{class } \mathbb{C}\langle \bar{\mathbb{X}}' \triangleleft \bar{\mathbb{T}}' \rangle \triangleleft \mathbb{N}' \{ \dots \}$ $\text{implies } i = j$ $\frac{\Delta_0 \vdash \mathbb{N} \text{ ok} \quad \Delta_0 \vdash \mathbb{N} <: \mathbb{N}_0}{\text{bind } \mathbb{N}_0 \bar{\mathbb{X}} = \mathbb{N}; \text{ ok}}_{[\text{TBind}]} \quad \frac{}{\text{module } \mathbb{D}\langle \bar{\mathbb{Y}} \triangleleft \bar{\mathbb{N}} \rangle \triangleleft \mathbb{N} \{ \bar{\mathbb{C}}\mathbb{L} \} \text{ ok}}_{[\text{TModule}]}$	

Table 3: Well-formed Declarations

method found in that supertype is the only method of that name that is statically guaranteed to exist. During computation, the annotated type is reduced whenever possible, modeling the downward search semantics of hygienic mixin method overriding.

Like receiver expressions, the arguments in a new expression are annotated with static types. These annotations are used at run-time to determine which constructor is referred to by the new operation. This is because the semantics require an exact match. There could be cases where multiple constructors match the

$\frac{\Delta; \Gamma \vdash \mathbf{x} \in \Gamma(\mathbf{x})_{[\text{TVar}]}}{\Delta; \Gamma \vdash \mathbf{e} \in \mathbf{S} \quad \Delta \vdash \text{T ok} \quad \text{[TUCast]}} \Delta; \Gamma \vdash (\text{T})\mathbf{e} \in \text{T}$ $\frac{\text{bound}_{\Delta}(\mathbb{V}) = \mathbb{D}\langle\bar{\mathbb{Z}}\rangle \quad \Delta; \Gamma \vdash \bar{\mathbf{e}} \in \bar{\mathbf{S}} \quad \vdash \mathbb{D}\langle\bar{\mathbb{Z}}\rangle.\mathbf{C}\langle\bar{\mathbf{T}}\rangle \text{ includes init}(\bar{\mathbf{S}})}{\Delta; \Gamma \vdash \text{new } \mathbb{V}.\mathbf{C}\langle\bar{\mathbf{T}}\rangle(\bar{\mathbf{e}}) \in \mathbb{D}\langle\bar{\mathbb{Z}}\rangle.\mathbf{C}\langle\bar{\mathbf{T}}\rangle \text{ annotate } [\bar{\mathbf{e}} :: \bar{\mathbf{S}}]} \text{[TNew]}$ $\frac{\Delta; \Gamma \vdash \mathbf{e} \in \text{T} \quad \Delta \vdash \text{T} <: \text{N} \quad \text{N} = \mathbb{V}.\mathbf{C}\langle\bar{\mathbf{U}}\rangle \quad \text{bound}_{\Delta}(\mathbb{V}) = \mathbb{D}\langle\bar{\mathbb{Z}}\rangle \quad \Delta \vdash \text{fields}(\text{N}) = \bar{\mathbf{T}} \bar{\mathbf{f}} \quad \Delta \vdash \text{P} <: \text{N} \text{ and } \mathbf{f}_i \in (\Delta \vdash \text{fields}(\text{P})) \text{ implies } \text{P} = \text{N}}{\Delta; \Gamma \vdash \mathbf{e}.\mathbf{f}_i \in \text{T}_i \text{ annotate } [\mathbf{e} :: \mathbb{D}\langle\bar{\mathbb{Z}}\rangle.\mathbf{C}\langle\bar{\mathbf{U}}\rangle]} \text{[TField]}$ $\frac{\Delta \vdash \bar{\mathbf{T}} \text{ ok} \quad \Delta; \Gamma \vdash \mathbf{e}_0 \in \text{T}_0 \quad \Delta; \Gamma \vdash \bar{\mathbf{e}} \in \bar{\mathbf{R}} \quad \text{bound}_{\Delta}(\text{T}_0) = \mathbb{D}\langle\bar{\mathbb{Z}}\rangle.\mathbf{C}\langle\bar{\mathbf{S}}\rangle \quad \Delta \vdash \text{mtype}(\text{m}, \mathbb{D}\langle\bar{\mathbb{Z}}\rangle.\mathbf{C}\langle\bar{\mathbf{S}}\rangle) = \text{P}.\langle\bar{\mathbf{X}} \triangleleft \bar{\mathbf{N}}\rangle \text{S m}(\bar{\mathbf{U}} \bar{\mathbf{x}}) \quad \Delta \vdash \bar{\mathbf{T}} <: [\bar{\mathbf{X}} \mapsto \bar{\mathbf{T}}]\bar{\mathbf{N}} \quad \Delta \vdash \bar{\mathbf{R}} <: [\bar{\mathbf{X}} \mapsto \bar{\mathbf{T}}]\bar{\mathbf{U}}}{\Delta; \Gamma \vdash \mathbf{e}_0.\text{m}\langle\bar{\mathbf{T}}\rangle(\bar{\mathbf{e}}) \in [\bar{\mathbf{X}} \mapsto \bar{\mathbf{T}}]\text{S} \text{ annotate } [\mathbf{e}_0 \in \text{P}]} \text{[TInv]}$	$\frac{\text{bound}_{\Delta}(\mathbb{V}) = \mathbb{D}\langle\bar{\mathbb{Z}}\rangle \quad \Delta \vdash \mathbb{D}\langle\bar{\mathbb{Z}}\rangle.\mathbf{C}\langle\bar{\mathbf{T}}\rangle \text{ ok} \quad \Delta; \Gamma \vdash \bar{\mathbf{e}} \in \bar{\mathbf{R}} \quad \Delta \vdash \bar{\mathbf{R}} <: \bar{\mathbf{S}} \quad \Delta \vdash \bar{\mathbf{S}} \text{ ok} \quad \vdash \mathbb{D}\langle\bar{\mathbb{Z}}\rangle.\mathbf{C}\langle\bar{\mathbf{T}}\rangle \text{ includes init}(\bar{\mathbf{S}})}{\Delta; \Gamma \vdash \text{new } \mathbb{V}.\mathbf{C}\langle\bar{\mathbf{T}}\rangle(\bar{\mathbf{e}} :: \bar{\mathbf{S}}) \in \mathbb{D}\langle\bar{\mathbb{Z}}\rangle.\mathbf{C}\langle\bar{\mathbf{T}}\rangle} \text{[TAnnNew]}$ $\frac{\Delta \vdash \text{fields}(\text{N}) = \bar{\mathbf{T}} \bar{\mathbf{f}} \quad \Delta \vdash \text{N ok} \quad \Delta; \Gamma \vdash \mathbf{e} \in \text{T} \quad \Delta \vdash \text{T} <: \text{N}}{\Delta; \Gamma \vdash [\mathbf{e} :: \text{N}].\mathbf{f}_i \in \text{T}_i} \text{[TAnnField]}$ $\frac{\Delta \vdash \bar{\mathbf{T}} \text{ ok} \quad \Delta; \Gamma \vdash \mathbf{e}_0 \in \text{T}_0 \quad \Delta; \Gamma \vdash \bar{\mathbf{e}} \in \bar{\mathbf{R}} \quad \Delta \vdash \text{mtype}(\text{m}, \mathbf{0}) = \text{P}.\langle\bar{\mathbf{X}} \triangleleft \bar{\mathbf{N}}\rangle \text{S m}(\bar{\mathbf{U}} \bar{\mathbf{x}}) \quad \Delta \vdash \bar{\mathbf{T}} <: [\bar{\mathbf{X}} \mapsto \bar{\mathbf{T}}]\bar{\mathbf{N}} \quad \Delta \vdash \bar{\mathbf{R}} <: [\bar{\mathbf{X}} \mapsto \bar{\mathbf{T}}]\bar{\mathbf{U}}}{\Delta; \Gamma \vdash [\mathbf{e}_0 \circ \mathbf{0}].\text{m}\langle\bar{\mathbf{T}}\rangle(\bar{\mathbf{e}}) \in [\bar{\mathbf{X}} \mapsto \bar{\mathbf{T}}]\text{S}} \text{[TAnnInv]}$
---	---

Table 4: Expression Typing

required signature of a new expression.

In order to allow for a subject-reduction theorem over the CCG small-step semantics, it is necessary to provide separate typing rules for annotated field lookup and method invocation expressions. Notice that it is not possible to simply ignore annotations during typing since accidental shadowing and overriding would cause the method and field types determined by the typing rules to change during computation. Just as type annotations play a crucial role in preserving information in the computation rules, they play an analogous role in typing expressions during computation.

In FGJ, “stupid casts” (the casting of an expression to an incompatible type) were identified as a possible result during subject reduction. In CCG, it is not possible to statically detect “stupid casts” in modules because class types cannot be completely resolved until module linking at run-time. For the sake of brevity, all casts are accepted during typing.

To avoid the complications of matching multiple constructors of an object, CCG requires an exact match between the parameter types of a constructor and the static types of the provided arguments. Casts can be inserted to coerce argument types.

2.8 Computation

The rules for Computation are contained in Figure 7. Computation is specified by a small-step semantics. The static type of a receiver is used to resolve method applications and field lookups, static types must be preserved during computation as annotations on receiver expressions. In contrast to FGJ and CMG, the small-step semantics for CGEN carry a bound environment Δ during computation representing the available bind declarations.

When evaluating the application of a method, the appropriate method body is found according to the mapping `mbody`. The application is then reduced to the body of the method, substituting all parameters with their instantiations, and `this` with the receiver. Because it is important that a method application is

$\frac{\mathbb{D}\langle\overline{Y}'\rangle.\mathbb{C}\langle\overline{T}'\rangle \text{ includes } \text{init}(\overline{T})}{\vdash \mathbb{D}\langle\overline{Y}'\rangle.\mathbb{C}\langle\overline{T}'\rangle \text{ provides } \mathbb{C}(\overline{T} \ \overline{x}) \ \{\dots\}}$ $\frac{\mathbb{D}\langle\overline{Y}'\rangle.\mathbb{C}\langle\overline{T}'\rangle \text{ includes } \langle\overline{Y} \triangleleft \overline{T}'\rangle \ \mathbb{R}' \ \mathbb{m}(\overline{U}' \ \overline{x})}{\vdash \mathbb{D}\langle\overline{Y}'\rangle.\mathbb{C}\langle\overline{T}'\rangle \text{ provides } \langle\overline{Y} \triangleleft \overline{T}'\rangle \ \mathbb{R}' \ \mathbb{m}(\overline{U}' \ \overline{x}) \ \{\text{return } \mathbf{e};\}}$
$MT(\mathbb{D}) = \text{module } \mathbb{D}\langle\overline{Y} \triangleleft \overline{N}\rangle \triangleleft \mathbb{N} \ \{\overline{\mathbb{C}\mathbb{L}}\} \text{ and}$ $\text{class } \mathbb{C}\langle\overline{X} \triangleleft \overline{S}\rangle \triangleleft \mathbb{U} \ \{\overline{\mathbb{T}} \ \overline{f}; \ \overline{\mathbb{K}} \ \overline{\mathbb{M}}\} \in \overline{\mathbb{C}\mathbb{L}} \text{ implies}$ $(\overline{S}', \mathbb{U}') = [\overline{Y} \mapsto \overline{Y}'][\overline{X} \mapsto \overline{X}'](\overline{S}, \mathbb{U}) \text{ and } [\overline{Y} \mapsto \overline{Y}'][\overline{X} \mapsto \overline{X}']\overline{\mathbb{T}} \ \overline{f}; \subseteq \overline{\mathbb{T}}' \ \overline{f}';$ $\text{and } \mathbb{D}\langle\overline{Y}'\rangle.\mathbb{C}\langle\overline{X}'\rangle \text{ provides } [\overline{Y} \mapsto \overline{Y}'][\overline{X} \mapsto \overline{X}'](\overline{\mathbb{K}}, \overline{\mathbb{M}})$ <hr/> $\Delta \vdash \text{classOverride}(\mathbb{D}\langle\overline{Y}'\rangle, \text{class } \mathbb{C}\langle\overline{X}' \triangleleft \overline{S}'\rangle \triangleleft \mathbb{U}' \ \{\overline{\mathbb{T}}' \ \overline{f}'; \ \overline{\mathbb{K}}' \ \overline{\mathbb{M}}'\})$

Table 6: Module Auxiliary Functions

not reduced until the most specific matching type annotation of the receiver is found, two separate forms are used for type annotations. The original type annotation marks the receiver with an annotation of the form $\in T$. This form of annotation is kept until no further reduction of the static type is possible. At this point, the form of the annotation is switched to $:: T$. Because the computation rules dictate that methods can be applied only on receivers whose annotations are of the latter form, we are ensured that no further reduction is possible when a method is applied. The symbol \circ is used to designate contexts where either form of annotation is applicable.

2.9 Type Soundness

In this section we establish a proof of type soundness for Core CGEN. The proof of type soundness for CCG is based on the proofs for type soundness for FGJ and CMG. Since CCG uses separate type variables for class and module parameterization, the proof contains many parallel lemmas showing key properties hold for each case.

Lemma 1 (Weakening). *Suppose $\Delta + \overline{X} \triangleleft \overline{N} \vdash \overline{N} \text{ ok}$, $\Delta + \overline{X} \triangleleft \overline{N} \vdash \overline{N} \text{ ok}$ and $\Delta \vdash \mathbb{U} \text{ ok}$.*

1. *If $\Delta \vdash \mathbb{S} <: \mathbb{T}$, then $\Delta + \overline{X} \triangleleft \overline{N} \vdash \mathbb{S} <: \mathbb{T}$ and $\Delta + \overline{X} \triangleleft \overline{N} \vdash \mathbb{S} <: \mathbb{T}$*
2. *If $\Delta \vdash \mathbb{S} <: \mathbb{T}$, then $\Delta + \overline{X} \triangleleft \overline{N} \vdash \mathbb{S} <: \mathbb{T}$ and $\Delta + \overline{X} \triangleleft \overline{N} \vdash \mathbb{S} <: \mathbb{T}$*
3. *If $\Delta \vdash \mathbb{S} \text{ ok}$, then $\Delta + \overline{X} \triangleleft \overline{N} \vdash \mathbb{S} \text{ ok}$ and $\Delta + \overline{X} \triangleleft \overline{N} \vdash \mathbb{S} \text{ ok}$.*
4. *If $\Delta \vdash \mathbb{S} \text{ ok}$, then $\Delta + \overline{X} \triangleleft \overline{N} \vdash \mathbb{S} \text{ ok}$ and $\Delta + \overline{X} \triangleleft \overline{N} \vdash \mathbb{S} \text{ ok}$*
5. *If $\Delta; \Gamma \vdash \mathbf{e} \in \mathbb{T}$, then*

$$\Delta; \Gamma, \mathbf{x} : \mathbb{U} \vdash \mathbf{e} \in \mathbb{T}, \Delta + \overline{X} \triangleleft \overline{N}; \Gamma \vdash \mathbf{e} \in \mathbb{T} \text{ and } \Delta + \overline{X} \triangleleft \overline{N}; \Gamma \vdash \mathbf{e} \in \mathbb{T}.$$

Proof. Each of the above cases is proved by straightforward induction on the derivation of the premise. \square

Lemma 2 (Class Type Substitution Preserves Fields). *For bound environment Δ s.t., types \overline{U} and non-variable class type \mathbb{N} , where $\overline{X} \notin \text{dom}(\Delta)$, if $\Delta \vdash \text{fields}(\mathbb{N}) = \overline{\mathbb{T}} \ \overline{f}$ then $\Delta \vdash \text{fields}([\overline{X} \mapsto \overline{U}]\mathbb{N}) = [\overline{X} \mapsto \overline{U}]\overline{\mathbb{T}} \ \overline{f}$*

$\frac{\Delta \vdash mbody(m\langle\bar{V}\rangle, N) = (\bar{x}, e_0)}{\Delta \vdash [\text{new } V.C\langle\bar{S}\rangle(\bar{e} :: P) :: N].m\langle\bar{V}\rangle(\bar{d}) \rightarrow [\bar{x} \mapsto \bar{d}][\text{this} \mapsto \text{new } V.C\langle\bar{S}\rangle(\bar{e} :: P)]e_0} \text{[RInv]}$	
$\begin{array}{l} \Delta \vdash e \in N \quad \Delta \vdash N <: V.C\langle\bar{U}\rangle \quad bound_{\Delta}(V) = \mathbb{D}\langle\bar{T}\rangle \\ MT(\mathbb{D}) = \text{module } \mathbb{D}\langle\bar{Y}\rangle \triangleleft \bar{N} \triangleleft N \{ \bar{C}\bar{L} \} \\ \text{class } C\langle\bar{X}\rangle \triangleleft \bar{S} \triangleleft T \{ \dots \} \in \bar{C}\bar{L} \\ \Delta \vdash mtype(m, \mathbb{D}\langle\bar{Z}\rangle.C\langle\bar{U}\rangle) = mtype(m, [\bar{Y} \mapsto \bar{Z}][\bar{X} \mapsto \bar{U}]T) \end{array}$	
$\frac{\Delta \vdash mtype(m, \mathbb{D}\langle\bar{Z}\rangle.C\langle\bar{U}\rangle) = mtype(m, [\bar{Y} \mapsto \bar{Z}][\bar{X} \mapsto \bar{U}]T)}{\Delta \vdash [e \in [\bar{Y} \mapsto \bar{Z}][\bar{X} \mapsto \bar{U}]T].m\langle\bar{V}\rangle(\bar{d}) \rightarrow [e \in \mathbb{D}\langle\bar{Z}\rangle.C\langle\bar{U}\rangle].m\langle\bar{V}\rangle(\bar{d})} \text{[RInvSub]}$	
$\begin{array}{l} \Delta \vdash e \in N \quad \Delta \vdash N <: V.C\langle\bar{U}\rangle \quad bound_{\Delta}(V) = \mathbb{D}\langle\bar{T}\rangle \\ MT(\mathbb{D}) = \text{module } \mathbb{D}\langle\bar{Y}\rangle \triangleleft \bar{N} \triangleleft N \{ \bar{C}\bar{L} \} \\ \text{class } C\langle\bar{X}\rangle \triangleleft \bar{S} \triangleleft T \{ \dots \} \in \bar{C}\bar{L} \\ \Delta \vdash mtype(m, \mathbb{D}\langle\bar{Z}\rangle.C\langle\bar{U}\rangle) \text{ is undefined or} \\ \Delta \vdash mtype(m, \mathbb{D}\langle\bar{Z}\rangle.C\langle\bar{U}\rangle) \neq mtype(m, [\bar{Y} \mapsto \bar{Z}][\bar{X} \mapsto \bar{U}]T) \end{array}$	
$\frac{\Delta \vdash mtype(m, \mathbb{D}\langle\bar{Z}\rangle.C\langle\bar{U}\rangle) \neq mtype(m, [\bar{Y} \mapsto \bar{Z}][\bar{X} \mapsto \bar{U}]T)}{\Delta \vdash [e \in [\bar{Y} \mapsto \bar{Z}][\bar{X} \mapsto \bar{U}]T].m\langle\bar{V}\rangle(\bar{d}) \rightarrow [e :: [\bar{Y} \mapsto \bar{Z}][\bar{X} \mapsto \bar{U}]T].m\langle\bar{V}\rangle(\bar{d})} \text{[RInvStop]}$	
$\frac{\Delta \vdash N <: T}{\Delta \vdash (T)\text{new } N(\bar{e} :: \bar{S}) \rightarrow \text{new } N(\bar{e} :: \bar{S})} \text{[RCast]}$	$\frac{\Delta \vdash fields(R) = \bar{T} \bar{f} \quad \Delta \vdash fieldVals(\text{new } N(\bar{e}), R) = \bar{e}'}{\Delta \vdash [\text{new } N(\bar{e}) :: R].f_i \rightarrow e'_i} \text{[RField]}$
<hr style="border: none; border-top: 1px solid black; margin: 10px 0;"/>	
$\frac{\Delta \vdash e_i \rightarrow e'_i}{\Delta \vdash \text{new } T(\dots, e_i :: S, \dots) \rightarrow \text{new } T(\dots, e'_i :: S, \dots)} \text{[RCNewArg]}$	
$\frac{\Delta \vdash e \rightarrow e'}{\Delta \vdash [e \circ N].m\langle\bar{V}\rangle(\bar{d}) \rightarrow [e' \circ N].m\langle\bar{V}\rangle(\bar{d})} \text{[RCInvRecv]}$	
$\frac{\Delta \vdash e_i \rightarrow e'_i}{\Delta \vdash [e \circ N].m\langle\bar{V}\rangle(\dots e_i \dots) \rightarrow [e \circ N].m\langle\bar{V}\rangle(\dots e'_i \dots)} \text{[RCInvArg]}$	
$\frac{\Delta \vdash e \rightarrow e'}{\Delta \vdash ((S)e) \rightarrow ((S)e')} \text{[RCCast]}$	$\frac{\Delta \vdash e \rightarrow e'}{\Delta \vdash [e :: R].f \rightarrow [e' :: R].f} \text{[RCField]}$

Table 7: Computation

Proof. Case analysis over the derivation of $\Delta \vdash fields(N) = \bar{T} \bar{f}$

Case $\Delta \vdash fields(\text{Object}) = \bullet$: Trivial.

Case $\Delta \vdash fields(V.C\langle\bar{S}\rangle) = [\bar{Y} \mapsto \bar{Z}][\bar{R} \mapsto \bar{S}]\bar{T} \bar{f}$: Assume $bound_{\Delta}(V) = \mathbb{D}\langle\bar{Z}\rangle$ and $MT(\mathbb{D}) = \text{module } \mathbb{D}\langle\bar{Y}\rangle \triangleleft \bar{N} \triangleleft N \{ \bar{C}\bar{L} \}$ and $\text{class } C\langle\bar{R}\rangle \triangleleft \bar{S}' \triangleleft T' \{ \dots \} \in \bar{C}\bar{L}$. We must show that $\Delta \vdash fields([\bar{X} \mapsto \bar{U}]\mathbb{D}\langle\bar{Z}\rangle.C\langle\bar{S}\rangle) = [\bar{X} \mapsto \bar{U}][\bar{Y} \mapsto \bar{Z}][\bar{R} \mapsto \bar{S}]\bar{T} \bar{f}$. Since $\bar{X} \notin dom(\Delta)$, we know $[\bar{X} \mapsto \bar{U}]\mathbb{D}\langle\bar{Z}\rangle.C\langle\bar{S}\rangle = \mathbb{D}\langle[\bar{X} \mapsto \bar{U}]\bar{Z}\rangle.C\langle[\bar{X} \mapsto \bar{U}]\bar{S}\rangle$. Then we have $fields(\mathbb{D}\langle[\bar{X} \mapsto \bar{U}]\bar{Z}\rangle.C\langle[\bar{X} \mapsto \bar{U}]\bar{S}\rangle) = [\bar{Y} \mapsto [\bar{X} \mapsto \bar{U}]\bar{Z}][\bar{R} \mapsto [\bar{X} \mapsto \bar{U}]\bar{S}]\bar{T} \bar{f}$. Since $\bar{X} \notin dom(\Delta)$, this simplifies to $[\bar{X} \mapsto \bar{U}][\bar{Y} \mapsto \bar{Z}][\bar{R} \mapsto \bar{S}]\bar{T} \bar{f}$. □

Lemma 3 (Module Type Substitution Preserves Fields). *For bound environment Δ , types \bar{U} and non-variable class type N , where $\bar{X} \notin dom(\Delta)$, if $\Delta \vdash fields(N) = \bar{T} \bar{f}$ then $\Delta \vdash fields([\bar{X} \mapsto \bar{U}]N) = [\bar{X} \mapsto \bar{U}]\bar{T} \bar{f}$*

Proof. Case analysis over the derivation of $\Delta \vdash \text{fields}(\mathbf{N}) = \bar{\mathbf{T}} \bar{\mathbf{f}}$

Case $\Delta \vdash \text{fields}(\text{Object}) = \bullet$: Trivial.

Case $\Delta \vdash \text{fields}(\mathbb{V}.C\langle\bar{\mathbf{S}}\rangle) = [\bar{\mathbf{Y}} \mapsto \bar{\mathbf{Z}}][\bar{\mathbf{R}} \mapsto \bar{\mathbf{S}}]\bar{\mathbf{T}} \bar{\mathbf{f}}$: Similiar to Lemma 2. □

Lemma 4 (Class Type Substitution Preserves Method Types). *For bound environment Δ , types $\bar{\mathbf{U}}$ and non-variable class type \mathbf{N} , where $\bar{\mathbf{X}} \notin \text{dom}(\Delta)$, if $\Delta \vdash \text{mtype}(\mathbf{m}, \mathbf{N}) = \mathbf{N}.<\bar{\mathbf{Y}} \triangleleft \bar{\mathbf{R}}> \mathbf{T} \mathbf{m}(\bar{\mathbf{U}}' \bar{\mathbf{x}})$ then $\Delta \vdash \text{mtype}(\mathbf{m}, [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\mathbf{N}) = [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}](\mathbf{N}.<\bar{\mathbf{Y}} \triangleleft \bar{\mathbf{R}}> \mathbf{T} \mathbf{m}(\bar{\mathbf{U}}' \bar{\mathbf{x}}))$.*

Proof. The premise $\Delta \vdash \text{mtype}(\mathbf{m}, \mathbf{N}) = \mathbf{N}.<\bar{\mathbf{Y}} \triangleleft \bar{\mathbf{R}}> \mathbf{T} \mathbf{m}(\bar{\mathbf{U}}' \bar{\mathbf{x}})$ matches only one of the two rules that defines *mtype*. And this rule applies equally well to the substituted forms. □

Lemma 5 (Module Type Substitution Preserves Method Types). *For bound environment Δ , types $\bar{\mathbf{U}}$ and non-variable class type \mathbf{N} , where $\bar{\mathbf{X}} \notin \text{dom}(\Delta)$, if $\Delta \vdash \text{mtype}(\mathbf{m}, \mathbf{N}) = \mathbf{N}.<\bar{\mathbf{Y}} \triangleleft \bar{\mathbf{R}}> \mathbf{T} \mathbf{m}(\bar{\mathbf{U}}' \bar{\mathbf{x}})$ then $\Delta \vdash \text{mtype}(\mathbf{m}, [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\mathbf{N}) = [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}](\mathbf{N}.<\bar{\mathbf{Y}} \triangleleft \bar{\mathbf{R}}> \mathbf{T} \mathbf{m}(\bar{\mathbf{U}}' \bar{\mathbf{x}}))$.*

Proof. Similar to Lemma 4. □

Lemma 6 (Class Type Substitution Preserves Constructor Inclusion). *For non-module types $\bar{\mathbf{U}}$ and non-variable class type \mathbf{N} , if $\vdash \mathbf{N}$ includes $\text{init}(\bar{\mathbf{S}})$ then $\vdash [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\mathbf{N}$ includes $[\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\text{init}(\bar{\mathbf{S}})$.*

Proof. Since we know $\mathbf{N} = \mathbb{D}\langle\bar{\mathbf{Z}}\rangle.C\langle\bar{\mathbf{R}}\rangle$, we have $\mathbb{D}\langle\bar{\mathbf{Z}}\rangle.C\langle\bar{\mathbf{R}}\rangle$ includes $[\bar{\mathbf{Y}} \mapsto \bar{\mathbf{Z}}][\bar{\mathbf{Y}} \mapsto \bar{\mathbf{R}}]\text{init}(\bar{\mathbf{S}})$. Because $[\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\mathbb{D}\langle\bar{\mathbf{Z}}\rangle.C\langle\bar{\mathbf{R}}\rangle = \mathbb{D}\langle[\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\bar{\mathbf{Z}}\rangle.C\langle[\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\bar{\mathbf{R}}\rangle$, we have $\mathbb{D}\langle[\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\bar{\mathbf{Z}}\rangle.C\langle[\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\bar{\mathbf{R}}\rangle$ includes $[\bar{\mathbf{Y}} \mapsto [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\bar{\mathbf{Z}}][\bar{\mathbf{Y}} \mapsto [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\bar{\mathbf{R}}]\text{init}(\bar{\mathbf{S}})$. But $[\bar{\mathbf{Y}} \mapsto [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\bar{\mathbf{Z}}][\bar{\mathbf{Y}} \mapsto [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\bar{\mathbf{R}}]\text{init}(\bar{\mathbf{S}}) = [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}][\bar{\mathbf{Y}} \mapsto \bar{\mathbf{Z}}][\bar{\mathbf{Y}} \mapsto \bar{\mathbf{R}}]\text{init}(\bar{\mathbf{S}})$, finishing the case. □

Lemma 7 (Module Type Substitution Preserves Constructor Inclusion). *For module types $\bar{\mathbf{U}}$ and non-variable class type \mathbf{N} , if $\vdash \mathbf{N}$ includes $\text{init}(\bar{\mathbf{S}})$ then $\vdash [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\mathbf{N}$ includes $[\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\text{init}(\bar{\mathbf{S}})$.*

Proof. Similar to Lemma 6. □

Lemma 8 (Class Type Substitution Preserves Method Inclusion). *For bound environment Δ , non-module types $\bar{\mathbf{U}}$ and non-variable class type \mathbf{N} , where $\bar{\mathbf{X}} \notin \text{dom}(\Delta)$, if $\vdash \mathbf{N}$ includes $\langle\bar{\mathbf{Y}} \triangleleft \bar{\mathbf{R}}\rangle \mathbf{T} \mathbf{m}(\bar{\mathbf{U}}' \bar{\mathbf{x}})$ then $\vdash [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\mathbf{N}$ includes $[\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}](\mathbf{N}.<\bar{\mathbf{Y}} \triangleleft \bar{\mathbf{R}}> \mathbf{T} \mathbf{m}(\bar{\mathbf{U}}' \bar{\mathbf{x}}))$.*

Proof. Similar to Lemma 6. □

Lemma 9 (Module Type Substitution Preserves Method Inclusion). *For bound environment Δ , module types $\bar{\mathbf{U}}$ and non-variable class type \mathbf{N} , where $\bar{\mathbf{X}} \notin \text{dom}(\Delta)$, if $\vdash \mathbf{N}$ includes $\langle\bar{\mathbf{Y}} \triangleleft \bar{\mathbf{R}}\rangle \mathbf{T} \mathbf{m}(\bar{\mathbf{U}}' \bar{\mathbf{x}})$ then $\vdash [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\mathbf{N}$ includes $[\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}](\mathbf{N}.<\bar{\mathbf{Y}} \triangleleft \bar{\mathbf{R}}> \mathbf{T} \mathbf{m}(\bar{\mathbf{U}}' \bar{\mathbf{x}}))$.*

Proof. Similar to Lemma 6. □

Lemma 10 (Class Type Substitution Preserves Class Subtyping). *For bound environment Δ and non-module types $\bar{\mathbf{U}}$, where $\bar{\mathbf{X}} \notin \text{dom}(\Delta)$, if $\Delta + \bar{\mathbf{X}} \triangleleft \bar{\mathbf{N}} \vdash \mathbf{S} <: \mathbf{T}$ and $\Delta \vdash \bar{\mathbf{U}} <: [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\bar{\mathbf{N}}$ then $\Delta \vdash [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\mathbf{S} <: [\bar{\mathbf{X}} \mapsto \bar{\mathbf{U}}]\mathbf{T}$.*

Proof. By structural induction over the derivation of $\Delta + \bar{\mathbf{X}} \triangleleft \bar{\mathbf{N}} \vdash \mathbf{S} <: \mathbf{T}$

Case SC-Reflex: Trivial.

Case SM-Reflex: Impossible since S is not a module type.

Case SC-Trans: Follows immediately from the induction hypothesis.

Case SM-Trans: Impossible since S is not a module type.

Case SC-Bound: $S = X$. If $X \in \text{dom}(\Delta)$, then it's trivial. On the other hand if $S = X_i$, $T = N_i$, $\Delta + \bar{X} \triangleleft \bar{N} \vdash S <: N_i$. Then $[\bar{X} \mapsto \bar{U}]S = U_i$ and $[\bar{X} \mapsto \bar{U}]T = [\bar{X} \mapsto \bar{U}]N_i$. But we're given that $\Delta \vdash \bar{U} <: [\bar{X} \mapsto \bar{U}]\bar{N}$. Finally Lemma 1 finishes the case.

Case SM-Bound: $S = \bar{X}$. Impossible since S is not a module type.

Case S-Class: $S = \mathbb{D} \langle \bar{Z} \rangle . \mathbb{C} \langle \bar{V} \rangle$ where $MT(\mathbb{D}) = \text{module } \mathbb{D} \langle \bar{Y} \triangleleft \bar{N} \rangle \triangleleft \mathbb{N} \{ \bar{C} \bar{L} \}$, $\text{class } \mathbb{C} \langle \bar{R} \triangleleft \bar{S} \rangle \triangleleft \mathbb{T}' \{ \dots \} \in \bar{C} \bar{L}$, and $[\bar{Y} \mapsto \bar{Z}][\bar{R} \mapsto \bar{V}]T' = T$. We must show that $\Delta \vdash [\bar{X} \mapsto \bar{U}]\mathbb{D} \langle \bar{Z} \rangle . \mathbb{C} \langle \bar{V} \rangle <: [\bar{X} \mapsto \bar{U}][\bar{Y} \mapsto \bar{Z}][\bar{R} \mapsto \bar{V}]T'$. But notice that $[\bar{X} \mapsto \bar{U}]\mathbb{D} \langle \bar{Z} \rangle . \mathbb{C} \langle \bar{V} \rangle = \mathbb{D} \langle [\bar{X} \mapsto \bar{U}]\bar{Z} \rangle . \mathbb{C} \langle [\bar{X} \mapsto \bar{U}]\bar{V} \rangle$. Next, by applying [S-Class], we can further reason $\Delta \vdash \mathbb{D} \langle [\bar{X} \mapsto \bar{U}]\bar{Z} \rangle . \mathbb{C} \langle [\bar{X} \mapsto \bar{U}]\bar{V} \rangle <: [\bar{Y} \mapsto [\bar{X} \mapsto \bar{U}]\bar{Z}][\bar{R} \mapsto [\bar{X} \mapsto \bar{U}]\bar{V}]T'$. But, $[\bar{Y} \mapsto [\bar{X} \mapsto \bar{U}]\bar{Z}][\bar{R} \mapsto [\bar{X} \mapsto \bar{U}]\bar{V}]T' = [\bar{X} \mapsto \bar{U}][\bar{Y} \mapsto \bar{Z}][\bar{R} \mapsto \bar{V}]T'$, finishing the case.

Case S-Module: $S = \mathbb{D} \langle \bar{Z} \rangle$. Impossible since S is not a module type. □

Lemma 11 (Module Type Substitution Preserves Class Subtyping). *For bound environment Δ and module types \bar{U} , where $\bar{X} \notin \text{dom}(\Delta)$, if $\Delta + \bar{X} \triangleleft \bar{N} \vdash S <: T$ and $\Delta \vdash \bar{U} <: [\bar{X} \mapsto \bar{U}]\bar{N}$ then $\Delta \vdash [\bar{X} \mapsto \bar{U}]S <: [\bar{X} \mapsto \bar{U}]T$.*

Proof. By structural induction over the derivation of $\Delta + \bar{X} \triangleleft \bar{N} \vdash S <: T$. The analysis is similar to Lemma 10. The only interesting case is:

Case SC-Bound: $S = X$. Since S is a class type we know $S \neq X_i$. Therefore, we have the trivial case $X \in \text{dom}(\Delta)$. □

Lemma 12 (Class Type Substitution Preserves Module Subtyping). *For bound environment Δ and non-module types \bar{U} , where $\bar{X} \notin \text{dom}(\Delta)$, if $\Delta + \bar{X} \triangleleft \bar{N} \vdash S <: T$ and $\Delta \vdash \bar{U} <: [\bar{X} \mapsto \bar{U}]\bar{N}$ then $\Delta \vdash [\bar{X} \mapsto \bar{U}]S <: [\bar{X} \mapsto \bar{U}]T$.*

Proof. By structural induction over the derivation of $\Delta + \bar{X} \triangleleft \bar{N} \vdash S <: T$. The analysis is similar to Lemma 10. The only interesting case is:

Case S-Module: $S = \mathbb{D} \langle \bar{Z} \rangle$, where $MT(\mathbb{D}) = \text{module } \mathbb{D} \langle \bar{Y} \triangleleft \bar{N} \rangle \triangleleft \mathbb{N} \{ \bar{C} \bar{L} \}$ and $[\bar{Y} \mapsto \bar{Z}]N = T$. We must show that $\Delta \vdash [\bar{X} \mapsto \bar{U}]\mathbb{D} \langle \bar{Z} \rangle <: [\bar{X} \mapsto \bar{U}][\bar{Y} \mapsto \bar{Z}]N$. But notice that $[\bar{X} \mapsto \bar{U}]\mathbb{D} \langle \bar{Z} \rangle = \mathbb{D} \langle [\bar{X} \mapsto \bar{U}]\bar{Z} \rangle$. Then by [S-Module], $\Delta \vdash \mathbb{D} \langle [\bar{X} \mapsto \bar{U}]\bar{Z} \rangle <: [\bar{Y} \mapsto [\bar{X} \mapsto \bar{U}]\bar{Z}]N$. But, $[\bar{Y} \mapsto [\bar{X} \mapsto \bar{U}]\bar{Z}]N = [\bar{X} \mapsto \bar{U}][\bar{Y} \mapsto \bar{Z}]N$, finishing this case. □

Lemma 13 (Module Type Substitution Preserves Module Subtyping). *For bound environment Δ and module types \bar{U} , where $\bar{X} \notin \text{dom}(\Delta)$, if $\Delta + \bar{X} \triangleleft \bar{N} \vdash S <: T$ and $\Delta \vdash \bar{U} <: [\bar{X} \mapsto \bar{U}]\bar{N}$ then $\Delta \vdash [\bar{X} \mapsto \bar{U}]S <: [\bar{X} \mapsto \bar{U}]T$.*

Proof. By structural induction over the derivation of $\Delta + \bar{X} \triangleleft \bar{N} \vdash S <: T$. The analysis is similar to Lemma 12. □

Lemma 14 (Class Type Substitution Preserves Class Well-formedness). *For bound environment Δ where $\bar{X} \notin \text{dom}(\Delta)$ and $\Delta \vdash \bar{U}$ ok, if $\Delta + \bar{X} \triangleleft \bar{N} \vdash S$ ok, $\Delta \vdash \bar{U} <: [\bar{X} \mapsto \bar{U}]\bar{N}$ then $\Delta \vdash [\bar{X} \mapsto \bar{U}]S$ ok.*

Proof. By structural induction over the derivation of $\Delta + \bar{X} \triangleleft \bar{N} \vdash S$ ok

Case WF-Object: Trivial.

Case WF-Mod: Trivial.

Case WF-Var: $\Delta + \bar{X} \triangleleft \bar{N} \vdash S$ ok. If $S \in \text{dom}(\Delta)$ this is trivial. Let $S = X_i$. Then $[\bar{X} \mapsto \bar{U}]S = U_i$. But we are give that $\Delta \vdash \bar{U}$ ok.

Case WF-MVar: Impossible since S is not a module type.

Case WF-Class: $\Delta + \bar{X} \triangleleft \bar{N} \vdash \mathbb{D} \langle \bar{T} \rangle . C \langle \bar{S} \rangle$ ok. Immediate from Lemma 10 and the induction hypothesis.

Case WF-Module: Impossible since S is not a module type. □

Lemma 15 (Module Type Substitution Preserves Class Well-formedness). *For bound environment Δ where $\bar{X} \notin \text{dom}(\Delta)$ and $\Delta \vdash \bar{U}$ ok, if $\Delta + \bar{X} \triangleleft \bar{N} \vdash S$ ok, $\Delta \vdash \bar{U} <: [\bar{X} \mapsto \bar{U}]\bar{N}$ then $\Delta \vdash [\bar{X} \mapsto \bar{U}]S$ ok.*

Proof. By structural induction over the derivation of $\Delta + \bar{X} \triangleleft \bar{N} \vdash S$ ok. Similar to Lemma 14. □

Lemma 16 (Class Type Substitution Preserves Module Well-formedness). *For bound environment Δ where $\bar{X} \notin \text{dom}(\Delta)$ and $\Delta \vdash \bar{U}$ ok, if $\Delta + \bar{X} \triangleleft \bar{N} \vdash S$ ok, $\Delta \vdash \bar{U} <: [\bar{X} \mapsto \bar{U}]\bar{N}$ then $\Delta \vdash [\bar{X} \mapsto \bar{U}]S$ ok.*

Proof. By structural induction over the derivation of $\Delta + \bar{X} \triangleleft \bar{N} \vdash S$ ok. Similar to Lemma 15. □

Lemma 17 (Module Type Substitution Preserves Module Well-formedness). *For bound environment Δ where $\bar{X} \notin \text{dom}(\Delta)$ and $\Delta \vdash \bar{U}$ ok, if $\Delta + \bar{X} \triangleleft \bar{N} \vdash S$ ok, $\Delta \vdash \bar{U} <: [\bar{X} \mapsto \bar{U}]\bar{N}$ then $\Delta \vdash [\bar{X} \mapsto \bar{U}]S$ ok.*

Proof. By structural induction over the derivation of $\Delta + \bar{X} \triangleleft \bar{N} \vdash S$ ok. The only interesting cases are:

Case WF-MVar: $\Delta + \bar{X} \triangleleft \bar{N} \vdash S$ ok. If $S \in \text{dom}(\Delta)$ this is trivial. Let $S = X_i$. Then $[\bar{X} \mapsto \bar{U}]S = U_i$. But we are give that $\Delta \vdash \bar{U}$ ok.

Case WF-Module: $\Delta + \bar{X} \triangleleft \bar{N} \vdash \mathbb{D} \langle \bar{T} \rangle$ ok. Immediate from Lemma 11 and the induction hypothesis. □

2.10 Module Hierarchies

We now build on our formalized notion of ground expressions and types and address the potential for cyclic and infinite hierarchies in CCG. We begin our analysis first with modules, and then in Section 2.11 we examine classes. The following lemmas are used to ensure that the constraints placed on module tables prevents cyclic and infinite hierarchies.

Lemma 18 (Module Compactness). *For bound environment Δ and module type $\mathbb{D}\langle\bar{\mathbb{N}}\rangle$ s.t. $\Delta \vdash \mathbb{D}\langle\bar{\mathbb{N}}\rangle$ ok, there is a finite chain of module types $\mathbb{P}_0, \dots, \mathbb{P}_N$ s.t. for all i s.t. $1 \leq i \leq N$, $\Delta \vdash \mathbb{P}_{i-1} <: \mathbb{P}_i$ and $\mathbb{P}_N = \text{Mod}$.*

Proof. A well-formed module table allows only one source for module parent types: modules can specify a specific module in its extends clause. Thus, this condition is required directly on all module instantiations for all well-formed module tables. \square

Lemma 19 (Antisymmetry). *For module types $\mathbb{C}\langle\bar{\mathbb{N}}\rangle, \mathbb{D}\langle\bar{\mathbb{P}}\rangle$, s.t. $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle$ ok, and $\Delta \vdash \mathbb{D}\langle\bar{\mathbb{P}}\rangle$ ok, if $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{D}\langle\bar{\mathbb{P}}\rangle$ then either $\Delta \not\vdash \mathbb{D}\langle\bar{\mathbb{P}}\rangle <: \mathbb{C}\langle\bar{\mathbb{N}}\rangle$ or $\mathbb{C}\langle\bar{\mathbb{N}}\rangle = \mathbb{D}\langle\bar{\mathbb{P}}\rangle$.*

Proof. By structural induction on the derivation of $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{D}\langle\bar{\mathbb{P}}\rangle$.

Case SM-Reflex: Then $\mathbb{C}\langle\bar{\mathbb{N}}\rangle = \mathbb{D}\langle\bar{\mathbb{P}}\rangle$.

Case S-Class: Impossible since $\mathbb{C}\langle\bar{\mathbb{N}}\rangle$ is a module type.

Case S-Module: Then $\mathbb{D}\langle\bar{\mathbb{P}}\rangle$ is a parent of $\mathbb{C}\langle\bar{\mathbb{T}}\rangle$. Therefore, the constraints on module heirarchy also require $\Delta \not\vdash \mathbb{D}\langle\bar{\mathbb{P}}\rangle <: \mathbb{C}\langle\bar{\mathbb{N}}\rangle$.

Case SM-Trans: In this case, there exists some \mathbb{T} s.t. $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{T}$ and $\Delta \vdash \mathbb{T} <: \mathbb{D}\langle\bar{\mathbb{P}}\rangle$. If $\mathbb{C}\langle\bar{\mathbb{N}}\rangle = \mathbb{D}\langle\bar{\mathbb{P}}\rangle$, we are done, so assume $\mathbb{C}\langle\bar{\mathbb{N}}\rangle \neq \mathbb{D}\langle\bar{\mathbb{P}}\rangle$. By the induction hypothesis, either $\mathbb{C}\langle\bar{\mathbb{N}}\rangle = \mathbb{T}$ or $\Delta \not\vdash \mathbb{T} <: \mathbb{C}\langle\bar{\mathbb{N}}\rangle$. But if $\mathbb{C}\langle\bar{\mathbb{N}}\rangle = \mathbb{T}$, then $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{D}\langle\bar{\mathbb{P}}\rangle$ was already derived as a premise to [SM-Trans], and then by the induction hypothesis, $\Delta \not\vdash \mathbb{D}\langle\bar{\mathbb{P}}\rangle <: \mathbb{C}\langle\bar{\mathbb{N}}\rangle$. Finally, consider the case that $\mathbb{C}\langle\bar{\mathbb{N}}\rangle \neq \mathbb{T}$. Then we can show that by contradiction, if $\Delta \vdash \mathbb{D}\langle\bar{\mathbb{P}}\rangle <: \mathbb{C}\langle\bar{\mathbb{N}}\rangle$. implies that $\Delta \vdash \mathbb{T} <: \mathbb{C}\langle\bar{\mathbb{N}}\rangle$ which contradicts the induction hypothesis. So $\Delta \not\vdash \mathbb{D}\langle\bar{\mathbb{P}}\rangle <: \mathbb{C}\langle\bar{\mathbb{N}}\rangle$. \square

Lemma 20 (Uniqueness). *For module type $\mathbb{C}\langle\bar{\mathbb{N}}\rangle$ s.t. $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle$ ok, there is exactly one type $\mathbb{P} \neq \mathbb{C}\langle\bar{\mathbb{N}}\rangle$ (i.e., the declared parent instantiation) s.t. both of the following conditions hold:*

1. $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{P}$
2. If $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$, $\mathbb{C}\langle\bar{\mathbb{N}}\rangle \neq \mathbb{O}$, and $\Delta \vdash \mathbb{O} <: \mathbb{P}$ then $\mathbb{O} = \mathbb{P}$.

Proof. Let \mathbb{P} be the declared parent instantiation of $\mathbb{C}\langle\bar{\mathbb{N}}\rangle$. Suppose for a contradiction that there was a type $\mathbb{O} \neq \mathbb{P}$ s.t. $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$, and $\Delta \vdash \mathbb{O} <: \mathbb{P}$. Then there is some finite derivation of $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$. Consider a shortest derivation of $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$, i.e., a derivation employing no more rule applications than any other derivation. Such a derivation can't conclude with [SM-Reflex] because $\mathbb{C}\langle\bar{\mathbb{N}}\rangle \neq \mathbb{O}$, nor [S-Class] because $\mathbb{C}\langle\bar{\mathbb{N}}\rangle$ is a module type. Also, it can't conclude with [S-Module] because $\mathbb{P} \neq \mathbb{O}$. Thus it must conclude with [SM-Trans]. Then there is some type \mathbb{O}' s.t. $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}'$ and $\Delta \vdash \mathbb{O}' <: \mathbb{O}$. Similarly, a shortest derivation of $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}'$ can't conclude with [SM-Reflex]; otherwise our derivation of $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$ is not the shortest derivation, nor [S-Class] since $\mathbb{C}\langle\bar{\mathbb{N}}\rangle$ is a module type. Our derivation of $\Delta \vdash \mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}'$ can't conclude with [S-Module]; otherwise $\mathbb{O}' = \mathbb{P}$, which is impossible by Lemma

19. Thus, a shortest derivation of $\Delta \vdash \mathbb{C}\langle\bar{N}\rangle <: \mathbb{O}'$ must conclude with [SM-Trans]. Continuing in this fashion, we can show that at each step in our derivation of $\Delta \vdash \mathbb{C}\langle\bar{N}\rangle <: \mathbb{O}$, the rule [SM-Trans] must be employed, requiring yet another step in the derivation. Thus, no finite length derivation could conclude with $\Delta \vdash \mathbb{C}\langle\bar{N}\rangle <: \mathbb{O}$, so $\Delta \not\vdash \mathbb{C}\langle\bar{N}\rangle <: \mathbb{O}$. \otimes Therefore, type \mathbb{O} does not exist. \square

2.11 Class Hierarchies

The presence of recursive modules in CGEN allows for the possibility of cyclic class hierarchies. We must ensure that the constraints placed on modules and bind declarations prevents cyclic class hierarchies from forming. We can guarantee the sanity of CCG class hierarchies with the following three lemmas:

Lemma 21 (Compactness). *For bound environment Δ and class type $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$ s.t. $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$ ok, there is a finite chain of class types P_0, \dots, P_N s.t. for all i s.t. $1 \leq i \leq N$, $\Delta \vdash P_{i-1} <: P_i$ and $P_N = \mathbf{Object}$.*

Proof. There are two sources for parent types for classes in a well-formed module table: (1) from either a locally defined class or (2) from a class introduced by a module import. We consider the later case to be a mixin type because its parent type is determined during module linking (instantiation).

In the case of non-mixin instantiations, the condition is required directly in the subclassing chain. In the case of mixin instantiations, we can show this condition holds through by contradiction. Assume there exists a mixin instantiation N where this required condition does not hold. This implies that the parent must be a mixin instantiation; otherwise the lemma would obviously be satisfied by the parent instantiation, and by [S-Class] for N as well. Lets call the parent type to be N' . By recursively following our argument, we can show that the parent type of N' should likewise be a mixin instantiation N'' , and so on. Recall that for an instantiated module $\mathbb{D}\langle\bar{T}\rangle$, the imported modules \bar{T} are determined with respect to a bound environment Δ . Since modules are linked through bind declarations, this implies the available bind declarations link a set of modules to create a cyclic class hierarchy. However, the restrictions on bind declarations, as discussed in Section 2.3 ensures that the set of linked modules generates an acyclic class hierarchy CT . \otimes Thus, the condition holds for all mixin instantiations as well as non-mixin class instantiations. \square

Lemma 22 (Antisymmetry). *For class types $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$, $\mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$ s.t. $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$ ok, $\Delta \vdash \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$ ok, if $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle <: \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$ then either $\Delta \not\vdash \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle <: \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$ or $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle = \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$.*

Proof. By structural induction on the derivation of $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle <: \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$. The interesting cases are:

Case SC-Reflex: Then $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle = \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$.

Case S-Class: Then $\mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$ is the parent of $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$.

Case SC-Trans: In this case, there exists some V s.t. $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle <: V$ and $\Delta \vdash V <: \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$. If $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle = \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$, we are done, so assume $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle \neq \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$. By the induction hypothesis, either $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle = V$ or $\Delta \not\vdash V <: \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$. But if $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle = V$, then $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle <: \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle$ was already derived as a premise to [SC-Trans], and then by the induction hypothesis, $\Delta \not\vdash \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle <: \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$. Finally, consider the case that $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle \neq V$. Then we can show that by contradiction, if $\Delta \vdash \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle <: \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$. implies that $\Delta \vdash V <: \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$ which contradicts the induction hypothesis. So $\Delta \not\vdash \mathbb{U}.\mathbb{D}\langle\bar{P}\rangle <: \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$. \square

Lemma 23 (Uniqueness). *For a given class type $\mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$ s.t. $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$ ok, there is exactly one type $P \neq \mathbb{T}.\mathbb{C}\langle\bar{N}\rangle$ (i.e., the declared parent instantiation) s.t. both of the following conditions hold:*

1. $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{P}$

2. If $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$, $\mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle \neq \mathbb{O}$, and $\Delta \vdash \mathbb{O} <: \mathbb{P}$ then $\mathbb{O} = \mathbb{P}$.

Proof. Let \mathbb{P} be the declared parent instantiation of $\mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle$. Suppose for a contradiction that there was a type $\mathbb{O} \neq \mathbb{P}$ s.t. $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$, and $\Delta \vdash \mathbb{O} <: \mathbb{P}$. Then there is some finite derivation of $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$. Consider a shortest derivation of $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$, i.e., a derivation employing no more rule applications than any other derivation. Such a derivation can't conclude with [SC-Reflex] because $\mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle \neq \mathbb{O}$. Also, it can't conclude with [S-Class] because $\mathbb{P} \neq \mathbb{O}$. It cannot conclude with [S-Module] since $\mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle$ is a class type. Thus it must conclude with [SC-Trans]. Then there is some type \mathbb{O}' s.t. $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}'$ and $\Delta \vdash \mathbb{O}' <: \mathbb{O}$. Similarly, a shortest derivation of $\vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}'$ can't conclude with [SC-Reflex]; otherwise our derivation of $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$ is not the shortest derivation. Also, our derivation of $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}'$ can't conclude with [S-Class]; otherwise $\mathbb{O}' = \mathbb{P}$, which is impossible by Lemma 22. Thus, a shortest derivation of $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}'$ must conclude with [SC-Trans]. Continuing in this fashion, we can show that at each step in our derivation of $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$, the rule [S-Trans] must be employed, requiring yet another step in the derivation. Thus, no finite length derivation could conclude with $\Delta \vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$, so $\Delta \not\vdash \mathbb{T}.\mathbb{C}\langle\bar{\mathbb{N}}\rangle <: \mathbb{O}$. Therefore, type \mathbb{O} does not exist. \square

2.12 Preservation

With these lemmas in hand, we now proceed to show that substitution preserves typing in CGEN.

Lemma 24 (Class Type Substitution Preserves Typing). *For bound environment Δ and annotated non-module types $\bar{\mathbb{U}}$ s.t. $\Delta \vdash \bar{\mathbb{U}}$ ok and $\bar{\mathbb{X}} \notin \Delta$, if $\Delta + \bar{\mathbb{X}} \triangleleft \bar{\mathbb{N}}; \Gamma \vdash \mathbf{e} \in \mathbb{S}$, $\Delta \vdash \bar{\mathbb{U}} \triangleleft [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\bar{\mathbb{N}}$ then $\Delta; [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\Gamma \vdash [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbf{e} \in [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbb{S}$.*

Proof. By structural induction over the derivation $\Delta + \bar{\mathbb{X}} \triangleleft \bar{\mathbb{N}}; \Gamma \vdash \mathbf{e} \in \mathbb{S}$.

Case T-Var: $\mathbf{e} = \mathbf{x}$, $\Delta + \bar{\mathbb{X}} \triangleleft \bar{\mathbb{N}}; \Gamma \vdash \mathbf{e} \in \Gamma(\mathbf{x})$. Then $\Delta; [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\Gamma \vdash \mathbf{x} \in [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\Gamma(\mathbf{x})$.

Case T-Cast: $\mathbf{e} = (\mathbb{S})\mathbf{e}'$ where $\Delta; \Gamma \vdash \mathbf{e}' \in \mathbb{T}$. By Lemma 14, $\Delta \vdash [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbb{S}$ ok. By the induction hypothesis $\Delta; [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\Gamma \vdash [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbf{e}' \in [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbb{T}$. Then by [T-Cast], $\Delta; [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\Gamma \vdash [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbf{e} \in [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbb{S}$.

Case T-Ann-New, T-Ann-Field: Similar. The antecedents of these rules apply to the substituted forms by straightforward application of the induction hypothesis, and supporting substitution lemmas.

Case T-Ann-Invk: $\mathbf{e} = [\mathbf{e}_0 \circ \mathbb{O}].\mathbf{m}\langle\bar{\mathbb{T}}\rangle(\bar{\mathbf{e}}) \in [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbb{S}$. All the antecedents in [T-Ann-Invk] except for the method substitution type apply by straightforward application of the induction hypothesis, and supporting substitution lemmas. So we need to show that:

$$\Delta \vdash \mathit{mtype}(\mathbf{m}, [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbb{O}) = [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbb{P}.\langle\bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}}\rangle \mathbb{T} \mathbf{m}(\bar{\mathbb{U}}' \bar{\mathbf{x}}).$$

There are two cases:

Subcase $\mathbb{O} = \mathbb{P}$: The by Lemma 4, $\Delta \vdash \mathit{mtype}(\mathbf{m}, [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbb{O}) = [\bar{\mathbb{X}} \mapsto \bar{\mathbb{U}}]\mathbb{O}.\langle\bar{\mathbb{X}} \triangleleft \bar{\mathbb{R}}\rangle \mathbb{T} \mathbf{m}(\bar{\mathbb{U}}' \bar{\mathbf{x}})$.

Subcase $\mathbb{O} \neq \mathbb{P}$: Because the annotated type of the receiver in the original invocation expression from which \mathbf{e} was reduced was determined by [T-Invk], it must have matched \mathbb{P} . The only reduction of the original expression which could have modified the annotated type is [R-Inv-Sub]. But the antecedents of [R-Inv-Sub] ensure that an annotated type \mathbb{S} is reduced to \mathbb{T} only if $\Delta \vdash \mathit{mtype}(\mathbf{m}, \mathbb{S}) = \mathit{mtype}(\mathbf{m}, \mathbb{T})$. Therefore, $\Delta \vdash \mathit{mtype}(\mathbf{m}, \mathbb{O}) = \mathit{mtype}(\mathbf{m}, \mathbb{P})$ and the case is finished by Lemma 4.

□

Lemma 25 (Module Type Substitution Preserves Typing). *For annotated types \bar{U} s.t. $\Delta \vdash \bar{U}$ ok and $\bar{X} \notin \text{dom}(\Delta)$, if $\Delta + \bar{X} \triangleleft \bar{N}; \Gamma \vdash \mathbf{e} \in \mathbf{S}$, $\Delta \vdash \bar{U} \triangleleft [\bar{X} \mapsto \bar{U}] \bar{N}$ then $\Delta; [\bar{X} \mapsto \bar{U}] \Gamma \vdash [\bar{X} \mapsto \bar{U}] \mathbf{e} \in [\bar{X} \mapsto \bar{U}] \mathbf{S}$.*

Proof. By structural induction over the derivation $\Delta + \bar{X} \triangleleft \bar{N}; \Gamma \vdash \mathbf{e} \in \mathbf{S}$. Similar to Lemma 24. □

Lemma 26 (Term Substitution Preserves Typing). *For annotated expression \mathbf{e} , annotated expressions $\bar{\mathbf{e}}$, and types \bar{T} s.t. $\Delta \vdash \bar{T}$ ok, if $\Delta; \bar{\mathbf{x}} : \bar{T} \vdash \mathbf{e} \in \mathbf{S}$ and $\Delta \vdash \bar{\mathbf{e}} \in \bar{\mathbf{R}}$ where $\Delta \vdash \bar{\mathbf{R}} <: \bar{T}$ then $\Delta \vdash [\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \mathbf{e} \in \mathbf{S}'$ where $\Delta \vdash \mathbf{S}' <: \mathbf{S}$.*

Proof. By structural induction over the derivation of $\Delta; \bar{\mathbf{x}} : \bar{T} \vdash \mathbf{e} \in \mathbf{S}$.

Case T-Var: $\mathbf{e} = x_i$. Then $\vdash [\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \mathbf{e} = e_i$. Since $\Delta \vdash e_i \in \mathbf{R}_i$, we have $\mathbf{S}' = \mathbf{R}_i$.

Case T-Cast: $\mathbf{e} = (\mathbf{S})\mathbf{e}'$. By the induction hypothesis, $\vdash [\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \mathbf{e}'$ is well-typed, so $\Delta \vdash [\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \mathbf{e} \in \mathbf{S}$.

Case T-Ann-New: $\mathbf{e} = \text{new } \mathbf{S}(\bar{\mathbf{e}}' :: \bar{\mathbf{R}})$. But $[\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \text{new } \mathbf{S}(\bar{\mathbf{e}}' :: \bar{\mathbf{R}}) = \text{new } \mathbf{S}([\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \bar{\mathbf{e}}' :: \bar{\mathbf{R}})$. By the induction hypothesis, $\Delta \vdash [\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \bar{\mathbf{e}}' \in \bar{\mathbf{R}}'$ where $\Delta \vdash \bar{\mathbf{R}}' <: \bar{\mathbf{R}}$. So, by [T-Ann-Field], $\Delta \vdash \text{new } \mathbf{S}([\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \bar{\mathbf{e}}' :: \bar{\mathbf{R}}) \in \mathbf{S}$.

Case T-Ann-Field: $\mathbf{e} = [\mathbf{e}' :: \mathbf{N}] . \mathbf{f}$. Term substitution does not effect the annotation \mathbf{N} or field \mathbf{f} . From the induction hypothesis we know $\Delta \vdash \mathbf{e}' \in \mathbf{N}'$ where $\Delta \vdash \mathbf{N}' <: \mathbf{N}$. So by [T-Ann-Field], $\Delta \vdash [\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \mathbf{e} \in \mathbf{S}$.

Case T-Ann-Invk: $\mathbf{e} = [\mathbf{e}_0 \circ \mathbf{O}] . \mathbf{m} \langle \bar{\mathbf{T}} \rangle (\bar{\mathbf{e}}')$. By the induction hypothesis, $[\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \mathbf{e}_0$ is well-typed, as well as $[\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \bar{\mathbf{e}}'$. The other premises of [T-Ann-Invk] are not affected by term substitution, and the static type of the invocation is determined solely by \mathbf{m} and the annotated type of the receiver, neither of which are modified by term substitution. So, by [T-Ann-Invk], $\Delta \vdash [\bar{\mathbf{x}} \mapsto \bar{\mathbf{e}}] \mathbf{e} \in \mathbf{S}$.

□

With these lemmas in hand, we are now in a position to establish a subject reduction theorem.

Theorem 1 (Subject Reduction). *If $\Delta \vdash \mathbf{e} \in \mathbf{T}$ and $\Delta \vdash \mathbf{e} \rightarrow \mathbf{e}'$ then $\Delta \vdash \mathbf{e}' \in \mathbf{S}$ where $\Delta \vdash \mathbf{S} <: \mathbf{T}$.*

Proof. By structural induction over the derivation of $\Delta \vdash \mathbf{e} \rightarrow \mathbf{e}'$.

Case R-Cast: $\mathbf{e} = (\mathbf{O}) \text{new } \mathbf{N}(\bar{\mathbf{e}} :: \bar{\mathbf{S}})$. By [T-Cast], $\Delta \vdash \mathbf{e} \in \mathbf{O}$. By [R-Cast], $\Delta \vdash \mathbf{N} <: \mathbf{O}$. Finally, by [T-Ann-New], $\Delta \vdash \text{new } \mathbf{N}(\bar{\mathbf{e}} :: \bar{\mathbf{S}}) \in \mathbf{N}$, which finishes the case.

Case R-Field: $\mathbf{e} = [\text{new } \mathbf{T}_0 . \mathbf{C} \langle \bar{\mathbf{T}} \rangle (\bar{\mathbf{e}}) :: \mathbb{D}' \langle \bar{\mathbf{V}}' \rangle . \mathbf{C}' \langle \bar{\mathbf{T}}' \rangle] . \mathbf{f}_i$. By [R-Field], $\mathbf{e}' = \Delta \vdash \text{fieldVals}(\text{new } \mathbf{T}_0 . \mathbf{C} \langle \bar{\mathbf{T}} \rangle (\bar{\mathbf{e}} :: \bar{\mathbf{R}}), \mathbb{D}' \langle \bar{\mathbf{V}}' \rangle . \mathbf{C}' \langle \bar{\mathbf{T}}' \rangle)_i$. Let $\Delta \vdash \text{fields}(\mathbb{D}' \langle \bar{\mathbf{V}}' \rangle . \mathbf{C}' \langle \bar{\mathbf{T}}' \rangle) = \bar{\mathbf{S}} \bar{\mathbf{f}}$ and $\text{bound}_\Delta(\mathbf{T}_0) = \mathbb{D} \langle \bar{\mathbf{V}} \rangle$. By [T-Ann-Field], $\Delta \vdash \mathbf{e} \in \mathbf{S}_i$. Let

$$\begin{aligned} MT(\mathbb{D}) &= \text{module } \mathbb{D} \langle \bar{\mathbf{Y}} \rangle \triangleleft \bar{\mathbf{N}} \rangle \triangleleft \mathbf{N} \{ \bar{\mathbf{C}} \mathbf{L} \} \\ \text{class } \mathbf{C} \langle \bar{\mathbf{X}} \rangle \triangleleft \bar{\mathbf{N}} \rangle \triangleleft \mathbf{N} \{ \dots \mathbf{C}(\bar{\mathbf{U}} \bar{\mathbf{x}}) \{ \dots \} \dots \} &\in \bar{\mathbf{C}} \mathbf{L}, \\ MT(\mathbb{D}') &= \text{module } \mathbb{D}' \langle \bar{\mathbf{Y}}' \rangle \triangleleft \bar{\mathbf{N}}' \rangle \triangleleft \mathbf{N}' \{ \bar{\mathbf{C}} \mathbf{L}' \} \\ \text{class } \mathbf{C}' \langle \bar{\mathbf{X}}' \rangle \triangleleft \bar{\mathbf{N}}' \rangle \triangleleft \mathbf{N}' \{ \dots \} &\in \bar{\mathbf{C}} \mathbf{L}', \end{aligned}$$

We show by induction over the derivation of $\mathbf{e}' = \Delta \vdash \text{fieldVals}(\text{new } \mathbf{T}_0 . \mathbf{C} \langle \bar{\mathbf{T}} \rangle (\bar{\mathbf{e}} :: \bar{\mathbf{R}}), \mathbb{D}' \langle \bar{\mathbf{V}}' \rangle . \mathbf{C}' \langle \bar{\mathbf{T}}' \rangle)_i$ that $\Delta \vdash \mathbf{S} <: \mathbf{S}_i$. There are two possibilities:

Subcase $\mathbb{D}\langle\bar{V}\rangle.\mathbb{C}\langle\bar{T}\rangle = \mathbb{D}'\langle\bar{V}'\rangle.\mathbb{C}'\langle\bar{T}'\rangle$: Because we are given that $\Delta \vdash \mathbf{e} \rightarrow \mathbf{e}'$, it must be the case that $\Delta \vdash \mathit{fieldVals}(\mathbf{new} \mathbb{T}_0.\mathbb{C}\langle\bar{T}\rangle(\bar{\mathbf{e}}), \mathbb{D}\langle\bar{V}\rangle.\mathbb{C}\langle\bar{T}\rangle)_i = [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{T}][\bar{x} \mapsto \bar{\mathbf{e}}]e'_i$ where $\bar{\mathbf{e}}'$ are the expressions assigned to the fields of class \mathbb{C} in the matching constructor. By [T-Constructor], we know that $\Delta + \bar{Y} \triangleleft \bar{N} + \mathbf{thisMod} \triangleleft \mathbb{D}\langle\bar{V}\rangle + \bar{X} \triangleleft \bar{N}; \bar{x} : \bar{T} \vdash e'_i \in \mathbb{S}'_i$ where $\mathbb{S}_i = [\bar{Y} \mapsto \bar{N}][\bar{X} \mapsto \bar{N}]\mathbb{S}'_i$. Since $\Delta \vdash \bar{T}$ ok, we know by Lemma 24, 25 and 26 that $\Delta \vdash [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{T}][\bar{x} \mapsto \bar{\mathbf{e}}]e'_i \in [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{T}]\mathbb{S}'_i$.

Subcase $\mathbb{D}\langle\bar{V}\rangle.\mathbb{C}\langle\bar{T}\rangle \neq \mathbb{D}'\langle\bar{V}'\rangle.\mathbb{C}'\langle\bar{T}'\rangle$: Then $\Delta \vdash \mathit{fieldVals}(\mathbf{new} \mathbb{T}_0.\mathbb{C}\langle\bar{T}\rangle(\bar{\mathbf{e}}), \mathbb{D}'\langle\bar{V}'\rangle.\mathbb{C}'\langle\bar{T}'\rangle) = \mathit{fieldVals}(\mathbf{new} [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{T}]\mathbb{N}([\bar{x} \mapsto \bar{\mathbf{e}}]e''), \mathbb{D}'\langle\bar{V}'\rangle.\mathbb{C}'\langle\bar{T}'\rangle)$, where e'' are the arguments passed in the super-constructor call within the matching constructor of \mathbb{C} . But by the induction hypothesis, $\Delta \vdash \mathit{fieldVals}(\mathbf{new} [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{T}]\mathbb{N}([\bar{x} \mapsto \bar{\mathbf{e}}]e'')) \in \mathbb{S}'$ where $\Delta \vdash \mathbb{S}'_i <: \mathbb{S}_i$, finishing the case.

Case R-Invk: $\mathbf{e} = [\mathbf{new} \mathbb{T}_0.\mathbb{C}\langle\bar{T}\rangle(\bar{\mathbf{e}}) :: \mathbb{P}].\mathbf{m}\langle\bar{U}\rangle(\bar{\mathbf{d}})$, $\Delta \vdash \mathit{mbody}(\mathbf{m}\langle\bar{U}\rangle, \mathbb{P}) = (\bar{x}, \mathbf{e}_0)$ where $\mathit{bound}_\Delta(\mathbb{T}_0) = \mathbb{D}\langle\bar{V}\rangle$. Let

```

 $\Delta \vdash \mathit{mtype}(\mathbf{m}, \mathbb{P}) = \mathbb{D}\langle\bar{V}\rangle.\mathbb{D}\langle\bar{R}\rangle. [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}]\langle\bar{X}' \triangleleft \bar{T}'\rangle \mathbb{S}' \mathbf{m}(\bar{U}' \bar{x})$ 
 $MT(\mathbb{D}) = \mathbf{module} \mathbb{D}\langle\bar{Y} \triangleleft \bar{N}\rangle \triangleleft \bar{N} \{ \bar{CL} \}$ 
 $\mathbf{class} \mathbb{D}\langle\bar{X} \triangleleft \bar{S}\rangle \triangleleft \bar{T} \{ \dots \} \in \bar{CL}$ 

```

By [T-Ann-Inv], $\Delta \vdash \mathbf{e} \in [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}][\bar{X}' \mapsto \bar{U}]\mathbb{S}'$. Let $\Delta_1 = \Delta + \bar{Y} \triangleleft \bar{V} + \mathbf{thisMod} \triangleleft \mathbb{D}\langle\bar{V}\rangle + \bar{X} \triangleleft \bar{R} + \bar{X}' \triangleleft \bar{U}$, and $\Gamma = \bar{x} : \bar{U}' + \mathbf{this} : \mathbb{D}\langle\bar{V}\rangle.\mathbb{D}\langle\bar{R}\rangle$. By [T-Method], $\Delta_1; \Gamma \vdash \mathbf{e}_0 \in \mathbb{S}''$ where $\Delta_1 \vdash \mathbb{S}'' <: \mathbb{S}'$. By Lemma 24 and 25, $\Delta_1; \Gamma \vdash [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}][\bar{X}' \mapsto \bar{U}]\mathbf{e}_0 \in [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}][\bar{X}' \mapsto \bar{U}]\mathbb{S}''$ and by Lemma 10 and 11, $\Delta_1 \vdash [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}][\bar{X}' \mapsto \bar{U}]\mathbb{S}'' <: [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}][\bar{X}' \mapsto \bar{U}]\mathbb{S}'$. Also, by [T-Ann-Invk], $\Delta \vdash \bar{\mathbf{e}} \in \bar{U}''$ where $\Delta \vdash \bar{U}'' <: [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}][\bar{X}' \mapsto \bar{U}]\bar{U}'$. Then by applying Lemma 26, $\Delta \vdash [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}][\bar{X}' \mapsto \bar{U}][\bar{x} \mapsto \bar{\mathbf{e}}][\mathbf{this} \mapsto \mathbf{new} \mathbb{T}_0.\mathbb{C}\langle\bar{T}\rangle(\bar{\mathbf{e}})]\mathbf{e}_0 \in \mathbb{S}'''$, where $\Delta \vdash \mathbb{S}''' <: [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}][\bar{X}' \mapsto \bar{U}]\mathbb{S}''$. Using the transitivity of subtyping, $\Delta \vdash \mathbb{S}''' <: [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}][\bar{X}' \mapsto \bar{U}]\mathbb{S}'$, finishing this case.

Case R-Inv-Sub: $\mathbf{e} = [e'' \in [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}]\mathbb{O}].\mathbf{m}\langle\bar{R}\rangle(\bar{\mathbf{d}})$. $\mathbf{e}' = [e'' \in \mathbb{D}\langle\bar{V}\rangle.\mathbb{C}\langle\bar{R}\rangle].\mathbf{m}\langle\bar{R}\rangle(\bar{\mathbf{d}})$. Let $\Delta \vdash \mathit{mtype}(\mathbf{m}, [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}]\mathbb{O}) = \mathbb{P}.\langle\bar{X}' \triangleleft \bar{T}'\rangle \mathbb{R} \mathbf{m}(\bar{U} \bar{x})$. By [T-Ann-Invk], $\Delta \vdash \mathbf{e} \in [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}][\bar{X}' \mapsto \bar{T}']\mathbb{R}$. But by [R-Inv-Sub], $\Delta \vdash \mathit{mtype}(\mathbf{m}, [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}]\mathbb{O}) = \mathit{mtype}(\mathbf{m}, \mathbb{D}\langle\bar{V}\rangle.\mathbb{C}\langle\bar{R}\rangle)$, finishing the case.

Case R-Stop: $\mathbf{e} = [e'' \in [\bar{X} \mapsto \bar{V}]\mathbb{O}].\mathbf{m}\langle\bar{V}\rangle(\bar{\mathbf{d}})$. This rule alters the type annotation for the receiver. However, since [T-Ann-Invk] works on either form of annotation, the type of the expression is preserved.

Case RC-Cast: $\mathbf{e} = (\mathbb{S})\mathbf{e}_0$, $\mathbf{e}' = (\mathbb{S})\mathbf{e}'_0$. By the induction hypothesis, \mathbf{e}' is well-typed, so the type of \mathbf{e}'_0 is \mathbb{S} by [T-Cast].

Case RC-Field: $\mathbf{e} = (\mathbf{e} :: \mathbb{N}).\mathbf{f}_i$. Because [T-Ann-Field] determines the field type based solely on the annotated static type (which is not altered by [RC-Field]), the type of \mathbf{e}' is identical to that of \mathbf{e} .

Case RC-New-Arg: $\mathbf{e} = \mathbf{new} \mathbb{T}(\bar{\mathbf{e}} :: \bar{\mathbb{S}})$. Let \mathbf{e}_i be the reduced subexpression of \mathbf{e} , and let \mathbf{e}_i reduce to \mathbf{e}'_i in \mathbf{e}' . Let $\Delta \vdash \mathbf{e}_i \in \mathbb{R}$. By the induction hypothesis, $\Delta \vdash \mathbf{e}'_i \in \mathbb{R}'$ where $\Delta \vdash \mathbb{R}' <: \mathbb{R}$. Then [T-Ann-New] applies just as well to \mathbf{e}' as to \mathbf{e} with the static type preserved.

Case RC-Inv-Recv, RC-Inv-Arg: $\mathbf{e} = [\mathbf{e}_0 \in \mathbb{N}].\mathbf{m}\langle\bar{V}\rangle(\bar{\mathbf{d}})$. Let \mathbf{e}_i be the reduced subexpression in \mathbf{e} , and let \mathbf{e}_i be reduced to \mathbf{e}'_i in \mathbf{e}' . In both of these cases, the induction hypothesis ensures that \mathbf{e}'_i satisfies the required properties of \mathbf{e}_i in [T-Ann-Invk]. But since the type determined by [T-Ann-Invk] depends solely on \mathbf{m} and \mathbb{N} , and neither \mathbf{m} nor \mathbb{N} is altered by these reductions, the static type is preserved.

□

Notice that the preservation theorem above (as well as the supporting lemmas) establish preservation for annotated terms. But since terms are not annotated until type checking, it is important to establish that the types of the annotated terms match their types before annotation. This property is established with the following two lemmas (the first is merely a small supporting lemma for the second).

Lemma 27 (Class Locations of Method Type Signatures). *For non-variable type $\mathbb{0}$ and environment Δ where $\Delta \vdash \mathbb{0}$ ok, if $mtype(\mathbf{m}, \mathbf{N}) = \mathbb{0} \cdot \langle \bar{X} \triangleleft \bar{T} \rangle \mathbf{R} \mathbf{m}(\bar{U} \bar{x})$ then for any type \mathbb{O}' s.t. $\Delta \vdash \mathbf{N} <: \mathbb{O}' <: \mathbb{0}$, if $\Delta \vdash mtype(\mathbf{m}, \mathbf{N}) = mtype(\mathbf{m}, \mathbb{O}')$ then $\mathbb{O}' = \mathbb{0}$, i.e., $\mathbb{0}$ is the closest superclass containing \mathbf{m} with a matching method signature.*

Proof. Trivial induction on the derivation of $mtype(\mathbf{m}, \mathbf{N}) = \mathbb{0} \cdot \langle \bar{X} \triangleleft \bar{T} \rangle \mathbf{R} \mathbf{m}(\bar{U} \bar{x})$ □

Theorem 2 (Preservation of Types Under Annotation). *For environments Δ, Γ ,*

1. *If $\Delta; \Gamma \vdash \mathbf{e} \cdot \mathbf{f}_i \in \mathbf{T}$ then $\Delta; \Gamma \vdash [\mathbf{e} :: \mathbf{N}] \cdot \mathbf{f}_i \in \mathbf{T}$.*
2. *If $\Delta; \Gamma \vdash \mathbf{new} \mathbf{R}(\bar{\mathbf{e}}) \in \mathbf{T}$ then $\Delta; \Gamma \vdash \mathbf{new} \mathbf{R}(\bar{\mathbf{e}} :: \bar{\mathbf{N}}) \in \mathbf{T}$.*
3. *If $\Delta; \Gamma \vdash \mathbf{e} \cdot \mathbf{m} \langle \bar{\mathbf{V}} \rangle (\bar{\mathbf{e}}) \in \mathbf{T}$ then $\Delta; \Gamma \vdash [\mathbf{e} \circ \mathbf{N}] \cdot \mathbf{m} \langle \bar{\mathbf{V}} \rangle (\bar{\mathbf{e}}) \in \mathbf{T}$.*

Proof. By analysis over the typing rules that generate annotations.

1. $\Delta; \Gamma \vdash \mathbf{e} \cdot \mathbf{f}_i \in \mathbf{T}$. The only distinction between the antecedents of [T-Field] and [T-Ann-Field] is that the type \mathbf{N} , which is the type used for field accesses, is explicitly determined using the receiver in [T-Field]. By using Lemma 23, we have the condition that there is no proper subtype \mathbf{P} of \mathbf{N} s.t., $\Delta \vdash \mathit{fields}(\mathbf{P})$ includes \mathbf{f}_i , thus ensuring \mathbf{N} is unique. This unique type is used to annotate the receiver, and therefore the same type referred to in [T-Ann-Field]. Thus, $\Delta; \Gamma \vdash [\mathbf{e} :: \mathbf{N}] \cdot \mathbf{f}_i \in \mathbf{T}$. Because none of the argument expressions have been reduced, their static types will match the annotated types exactly and $\Delta \vdash \bar{\mathbf{e}} \in \bar{\mathbf{N}}$. The other antecedents of [T-New] match antecedents of [T-Ann-New] exactly.
2. $\Delta; \Gamma \vdash \mathbf{new} \mathbf{R}(\bar{\mathbf{e}}) \in \mathbf{T}$. Because none of the argument expressions have been reduced, their static types will match the annotated types exactly with $\Delta \vdash \bar{\mathbf{e}} \in \bar{\mathbf{N}}$. The other antecedents of [T-New] match antecedents of [T-Ann-New] exactly.
3. $\Delta; \Gamma \vdash \mathbf{e} \cdot \mathbf{m} \langle \bar{\mathbf{V}} \rangle (\bar{\mathbf{e}})$. Again, no reduction has occurred, so by Lemma 27 the annotated type $\mathbb{0}$ will match the closest supertype of the bound of the the static type \mathbf{T}_0 of the receiver that contains \mathbf{m} . Then $\Delta \vdash mtype(\mathbf{m}, \mathbb{0}) = mtype(\mathbf{m}, \mathit{bound}_\Delta(\mathbf{T}_0))$ and the case is finished by [T-Ann-Invk].

□

2.13 Progress

Lemma 28 (Field Values). *For bound environment Δ , non-variable type \mathbf{N} , If $\Delta \vdash \mathit{fields}(\mathbf{N}) = \bar{\mathbf{T}} \bar{\mathbf{f}}$ and $\Delta \vdash \mathbf{new} \mathbf{P}(\bar{\mathbf{e}} :: \bar{\mathbf{R}}) \in \mathbf{P}$ where $\Delta \vdash \mathbf{P} <: \mathbf{N}$ then $\Delta \vdash \mathit{fieldVals}(\mathbf{new} \mathbf{P}(\bar{\mathbf{e}} :: \bar{\mathbf{R}}), \mathbf{N}) = \bar{\mathbf{e}}'$ where $|\bar{\mathbf{e}}'| = |\bar{\mathbf{f}}|$.*

Proof. Induction over the derivation of $\mathit{fields}(\mathbf{N}) = \bar{\mathbf{T}} \bar{\mathbf{f}}$.

Case $N = \text{Object}$: The definition of *includes* specifies that `Object` includes only the zero-ary constructor. Since the rules of subtyping specify that `Object` is a subtype of only itself, $\Delta \vdash \text{fields}(\text{Object}) = \text{fieldVals}(\text{new Object}(), \text{Object}) = \bullet$.

Case $N = \text{T.C}\langle\bar{R}\rangle$, $\text{fields}(N) = [\bar{Y} \mapsto \bar{V}][\bar{X} \mapsto \bar{R}]\bar{T} \bar{f}$: We proceed by structural induction on the derivation of $\Delta \vdash P <: N$. The relevant cases are:

Subcase SC-Reflex: By [T-Constructor], every valid constructor in a class must initialize all fields \bar{f} with expressions \bar{e} .

Subcase SC-Trans: Follows immediately from the induction hypothesis.

Subcase SC-Bound: Impossible since this theorem applies only to non-variable type.

Subcase S-Class: N is the instantiated parent class of P . In this case, only one definition of *fieldVals* applies. So we know that $\Delta \vdash \text{fieldVals}(\text{new } P(\bar{e} :: \bar{S}), N) = \text{fieldVals}(\text{new } N(\bar{e}''), N) = \bar{e}'$ where \bar{e}'' is a subset of \bar{e} used in the super call. By reasoning analogous to case [S-Reflect], we know $|\bar{e}'| = |\bar{f}|$ (note that we cannot employ the induction hypothesis directly since the induction is over the derivation of $\Delta \vdash P <: N$, not the derivation of *fieldVals*).

□

Lemma 29 (Method Bodies). *If $\Delta \vdash \text{mtype}(m, N) = P.\langle\bar{X} \triangleleft \bar{T}\rangle R m(\bar{U} \bar{x})$ and $\Delta \vdash \bar{V} <: [\bar{X} \mapsto \bar{V}]\bar{T}$ then there exists some e s.t., $\Delta \vdash \text{mbody}(m\langle\bar{V}\rangle, N) = (\bar{x}, e)$.*

Proof. Trivial induction over the derivation of $\Delta \vdash \text{mtype}(m, N) = P.\langle\bar{X} \triangleleft \bar{T}\rangle R m(\bar{U} \bar{x})$. □

With these lemmas, we now proceed to define a progress theorem for CGEN. The theorem relies on the following definitions.

Definition 1 (Value). *A well-typed expression e is a **value** iff e is of the form $\text{new } \text{T.C}\langle\bar{T}\rangle(\bar{e})$ where $\text{bound}_\Delta(\text{T}) = \mathbb{D}\langle\bar{V}\rangle$ and all \bar{e} are values.*

Definition 2 (Bad Cast). *A well-typed expression e is a **bad cast** iff e is of the form $(\text{T})e'$ where $\Delta \vdash e' \in S$ and $\Delta \not\vdash S <: \text{T}$.*

Notice that bad casts include both “stupid casts” (in the parlance of FGJ) and invalid upcasts. Now let $\xrightarrow{*}$ be the transitive closure of the reduction relation \rightarrow . Then we can state a progress theorem for CCG as follows:

Theorem 3 (Progress). *For program (MT, BT, e) s.t. $\Delta_{BT} \vdash e \in R$, if $\Delta_{BT} \vdash e \xrightarrow{*} e'$ then either e' is a value, e' contains a bad cast, or there exists e'' s.t. $\Delta_{BT} \vdash e' \rightarrow e''$.*

Proof. Because $\Delta_{BT} \vdash e \xrightarrow{*} e'$ we know that e' is well-typed. We proceed by structural induction over the form of e' .

Case $e' = [\text{new } N(\bar{e} :: \bar{S}) :: P].f$: We know that by [T-Ann-Field] that $\Delta_{BT} \vdash \text{fields}(P) = \bar{T} \bar{f}$ where $f = f_i$. By Lemma 28, $\Delta_{BT} \vdash \text{fieldVals}(\text{new } N(\bar{e}), P) = \bar{e}''$ and $|\bar{e}''| = |\bar{f}|$. Then by [R-Field], $\Delta \vdash e' \rightarrow e''_i$.

Case $e' = [d :: P].f$, d is not a new expression: By [T-Ann-Invk], d is well-typed, so by the induction hypothesis, either d is a value, d contains a bad cast, or there exists a d' s.t., $\Delta_b dd \vdash d \rightarrow d'$. But since d is not a `new` expression, it can't be a value. If it contains a bad cast, then so does e' and we are done. And if $\Delta_{BT} \vdash d \rightarrow d'$ then by [RC-Field], $\Delta_{BT} \vdash [d :: P].f \rightarrow [d' :: P].f$.

Case $e' = [d \circ P].m\langle\bar{T}\rangle(\bar{e})$, d is not a new expression: Analogous to the case above. Case $e' = [\text{new } N(e) :: P].m\langle\bar{T}\rangle(\bar{e})$. By [T-Ann-Invk], $\Delta_{BT} \vdash \text{mtype}(m, P) = 0.\langle\bar{X}\rangle\langle\bar{T}\rangle R m(\bar{U} \bar{x})$ Then by Lemma 29, $\Delta_{BT} \vdash \text{mbody}(m, P) = (x, e'')$, and by [R-Invk], $\Delta_{BT} \vdash e' \rightarrow e''$.

Case $e' = [\text{new } N(e) \in P].m\langle\bar{T}\rangle(\bar{e})$: By [T-Ann-Invk] and [T-Ann-New], $\Delta_{BT} \vdash \text{new } N(\bar{e}) \in N$. By Theorem 2, $\Delta_{BT} \vdash N <: P$. If $\Delta_{BT} \vdash \text{mtype}(m, N) = \text{mtype}(m, P)$ then $\Delta_{BT} \vdash e' \rightarrow [\text{new } N(\bar{e}) \in N].m\langle\bar{T}\rangle(\bar{e})$ by [R-Invk-Sub]. Otherwise $\Delta_{BT} \vdash e' \rightarrow [\text{new } N(\bar{e}) :: P].m\langle\bar{T}\rangle(\bar{e})$ by [R-Invk-Stop], finishing the case.

Case $e' = \text{new } N(\bar{e} :: \bar{T})$: Then either e' is a value and we are finished, or there is some e_i in \bar{e} that is not a value. Then by the induction hypothesis, either e_i contains a bad cast (and then so does e'), or there exists some e'_i s.t. $\Delta_{BT} \vdash e_i \rightarrow e'_i$. Then by [RC-New-Arg], $\Delta_{BT} \vdash \text{new } N(\bar{e} :: \bar{T}) \rightarrow \text{new } N(e_0 :: T_0, \dots, e'_i :: T_i, \dots, e_N :: T_N)$, finishing the case.

Case $e' = (N)e''$: Because e' is well typed, we know there is some P s.t., $\Delta_{BT} \vdash e'' \in P$. If $\Delta_{BT} \not\vdash P <: N$ then e' is a bad cast and we are done. Otherwise $\Delta_{BT} \vdash e \rightarrow e''$ by [R-Cast].

□

2.14 Type Soundness

Theorem 4 (Type Soundness). *For program (MT, BT, e) s.t. $\Delta_{BT} \vdash e \in T$, evaluation of (MT, BT, e) yields one of the following results:*

1. $\Delta_{BT} \vdash e \xrightarrow{*} v$ where v is a value of type S and $\Delta_{BT} \vdash S <: T$.
2. $\Delta_{BT} \vdash e \xrightarrow{*} e'$ where e' contains a bad cast,
3. Evaluation never terminates, i.e., for every e' s.t. $\Delta_{BT} \vdash e \xrightarrow{*} e'$ there exists e'' s.t. $\Delta_{BT} \vdash e' \rightarrow e''$.

Proof. Immediate from Theorems 1, 2, and 3. □

References

- [1] E. Allen, J. Bannet, and R. Cartwright. First-class genericity for Java. In *OOPSLA*, 2003.
- [2] E. Allen, J. Bannet, and R. Cartwright. Mixins in Generic Java are sound. Technical report, Rice University, 2003.
- [3] A. Igarashi, B. Pierce, and P. Wadler. Featherweight Java: A minimal core calculus for Java and GJ. In *OOPSLA*, 1999.
- [4] J. Sasitorn and R. Cartwright. Efficient first-class generics on stock Java virtual machines. In *SAC*, 2006.
- [5] J. Sasitorn and R. Cartwright. Component NextGen: A sound and expressive component framework for Java. In *OOPSLA*, 2007.
- [6] J. Sasitorn and R. Cartwright. Deriving components from genericity. In *SAC*, 2007.