Fundamenta Informaticae 109 (2011) 237–253
DOI 10.3233/FI-2011-508
IOS Press

Properties of Java Simple Closures

Marco Bellia and M. Eugenia Occhiuto
Dipartimento di Informatica
Università di Pisa
Largo B. Pontecorvo, 3, I-56127 Pisa, Italy
bellia@di.unipi.it ; occhiuto@di.unipi.it

Abstract. In the last years, the Java community has been arguing about adding closures to Java in order to improve expressivity. The debate has not yet terminated but all proposals seem to converge towards a notion of Simple Closures which contain only the essential features of anonymous functions. This paper addresses the problem of defining a rigorous semantics for Simple Closures. The technique adopted is well known and has already been used to prove interesting properties of other extensions of Java. A minimal calculus is defined: Featherweight Java extended with Simple Closures. Syntax and semantics of such a calculus are defined and type safety, backward compatibility, and the abstraction property are proved.

1. Introduction

In the last few years extensions to Java focus on higher order mechanisms to enhance expressivity, conciseness, good structuring, reusability, and factoring of code [22, 20, 25, 11, 21, 5, 6, 9, 14, 26]. Proposals to add closures in Java have been discussed since 2006 [19, 12, 2]. Recent revisions [15, 10] of the proposals agree on several aspects and lead to a simplified structure of closures [24] that is illustrated by Mark Reynholds in his Straw-man proposal [23] and is at the basis of the current version of the JLS draft for JDK7 [3, 4]. Accordingly, a closure is a value that abstracts an arbitrary Java code and makes such a value available for assignment, parameter transmission, value returning and invocation. Although JLS draft is rather precise as far as the syntactic structure and the restrictions about the combination of closures with other Java constructs, it fails (together with all the above cited proposals) to provide a formal and rigorous semantics. Furthermore, some features currently considered in the other proposals are still under investigation for inclusion in JDK7. The lack of a formal semantics makes difficult *i)* to evaluate significance and compatibility of the new features, *ii)* to compare different features that are sharing same

Address for correspondence: Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo, 3, I-56127 Pisa, Italy

aims, and *iii)* to prove that any future implementation is correct (i.e. hits correctly the design aims). Such problems are addressed in this paper, resorting to a minimal calculus: Featherweight Java (FJ for short), to *i)* formalize semantics, and *ii)* study properties of Simple Closures and some interesting variants of them.

FJ was presented at the 1999 ACM Symposium [17] as a minimal core calculus for a formal model to study design properties of Java. The idea is to omit most of the concrete features of the full Java to concentrate on a small core language, fully significant, for all the relevant aspects of the properties under investigation. Initially, it was introduced to design and formalize the generic type system of Java [8] and prove its soundness [18]. More recently [16], it has been used *i)* to define a reduction semantics of inner classes in Java, *ii)* to investigate on properties of inner classes, including abstraction features and interaction with inheritance, and *iii)* to provide for compilation issues of inner classes.

In this paper, we start using the variant of FJ, called FGJ, which deals with generic types and extends it to model all essential features of Java that are involved in the properties of closures we investigate in this paper, namely typing and abstraction. The omitted features include side effects (sequencing, assignments and threads) and exceptions. The new calculus Featherweight Generic Java with Closures is here called FGCJ. The structure of closure, we consider, is the one of Simple Closure [24, 23, 3], hereafter called closure, in the form of *Expression Lambda* [15], i.e. having expressions as body, since blocks are not allowed in FGCJ. Accordingly, closures *i)* have types, *function types*, which extend the Java type system, *ii)* are first class values which can be bound to parameters, hence passed to methods or other closures, *iii)* are invoked receiving a complete list of expressions for the arguments (n expressions for n-arguments closure), i.e. no currying is admitted, *iv)* invocation always returns a value, *v)* use only *effectively-final* variables, i.e. single assignable variables.

The reduction semantics is used to prove three fundamental properties which are *type soundness*, which asserts consistency between the type system and computation, *backward compatibility* which asserts the consistency between old rules system and the extended one, *abstraction property* which asserts the semantics analog of $\beta-$conversion.

The paper is organized as follows: Section 2 resumes FGJ features, Section 3 defines syntax and semantics of FGCJ, Section 4 states and prove the properties, eventually Section 5 concludes the paper and Appendix A contains proof details.

2. Featherweight Java

A program in FJ (FGJ) [17] consists of a declaration of generic class definitions and of an expression to be evaluated using the class definitions. The expression corresponds to the body of the 0-arguments main method of Java. Here is a declaration for some typical class definitions in FGJ.

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

A complete definition of the syntax of FGJ consists of the grammar rules in **Table 1** that are labelled by the defined grammatical category indexed by FGJ. Symbol $\lhd$ is a notational shorthand for Java keyword `extends`. For syntactic regularity, (a) classes always specify the super class, possibly `Object`, and have exactly one constructor definition; (b) class constructors have one parameter for each class field with the same name as the field, invoke the super constructor on the fields of the super class and initialize the remaining fields to the corresponding parameters; (c) field access always specifies the receiver (object), possibly `this`. This results in the stylized form of the constructors in the example above. Both classes and methods may have generic type parameters. In the example, `X` and `Y` are type parameters of the class `Pair`, while `Z` is type parameter of method `setfst`. Each type parameter has a *bound*, in the example `Object` for all.

FGJ has no side effects. Hence, *sequencing* and *assignment* are strictly confined to constructor bodies. In particular, method bodies have always the form `return`, followed by an expression, as in the body of `setfst` in the example. The lack of Java constructs for sequencing control and for store updating (along with that of concurrency, and reflection) is the main advantage of the calculus in studying language properties that are not affected by side effects. In this way the calculus is, as much as possible, compact and takes advantage of the referential transparency. The latter one provides a simple reduction semantics which is crucial for rigorous, easy to derive, proofs of the language properties [13]. About compactness, FGJ has only five forms of expressions: One for *Object Creation*, as `new Pair(...)` in the body of `setfst` in the example, another for variables (namely, *parameter naming*), as `newfst` and `this`, one for *field access*, as `this.snd` always in the body of `setfst`. The remaining two forms are *method invocation* and *cast* as in the expression below.

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

The presence of *cast* in FGJ is justified from its fundamental role in compiling generic classes and could be ruled out of FGCJ. From a syntactic point of view, `this` is a keyword in Java, and is a variable in FGJ, however, in both languages, it has the semantics of *object self-reference*, see rule `GR-Invk`. We conclude this presentation considering the twofold role of referential transparency: first evaluation is entirely formalized within the syntax of FGJ (hence, the evaluation process results in a sequence of FGJ expressions reducing the first one to the last one, if any, which represents an error or its value), second the order in which expressions are reduced, if more than one can be selected, does not affect the final result. The reduction semantics of FGJ consists of the first three rules that appear in **Table 2: Computation** that deal with term evaluation, and of the first five rules in **Table 2: Congruence** that deal with redex selection. The remaining 18 rules of the semantics of FGJ deal with the type system and with term well-formedness. The rules of FGJ have labels that are indexed by FGJ in **Table 2, 4, 5**. As an example of computation, expression `e`, evaluated in the context of the declaration of the classes `Pair`, `A` and `B` (namely, the evaluation of the FGJ program constituted by the declaration plus expression `e`), results in the sequence:

$$\begin{array}{ll}
\texttt{(}\underline{\texttt{(Pair) new Pair<A,B>(new A(),new B())}}\texttt{).setfst<B>(new B())} & \\
\underline{\texttt{(new Pair<A,B>(new A(),new B())).setfst<B>(new B())}} & \texttt{by GR-C}\textsc{ast}_{\texttt{FGJ}} \\
\texttt{new Pair<B,B>(new B(),new B())} & \texttt{by GR-I}\textsc{nvk}_{\texttt{FGJ}}
\end{array}$$

3. Featherweight GCJ

3.1. Notation and General Conventions

In this paper we adopt the notation used in [18]: Accordingly, $\overline{\mathtt{f}}$ is a shorthand for a possibly empty sequence $\mathtt{f}_1, \ldots, \mathtt{f}_n$ (and similarly for $\overline{\mathtt{T}}, \overline{\mathtt{x}}$, etc.) and $\overline{\mathtt{M}}$ is a shorthand for $\mathtt{M}_1 \ldots \mathtt{M}_n$ (with no commas) where n is the size $|\overline{\mathtt{f}}|$, respectively $|\overline{\mathtt{M}}|$, i.e. the number of terms of the sequence. The empty sequence is $\circ$ and symbol "," denotes concatenation of sequences. Operations on pairs of sequences are abbreviated in the obvious way: $\overline{\mathtt{C}}\ \overline{\mathtt{f}}$ is $\mathtt{C}_1\ \mathtt{f}_1, \ldots, \mathtt{C}_n\ \mathtt{f}_n$ and similarly $\overline{\mathtt{C}}\ \overline{\mathtt{f}};$ is $\mathtt{C}_1\ \mathtt{f}_1; \ldots \mathtt{C}_n\ \mathtt{f}_n;$ and $\mathtt{this}.\overline{\mathtt{f}} = \overline{\mathtt{f}};$ is a shorthand for $\mathtt{this}.\mathtt{f}_1 = \mathtt{f}_1; \ldots \mathtt{this}.\mathtt{f}_n = \mathtt{f}_n;$ Sequences of field declarations, parameters and method declaration cannot contain duplications. Cast, $(\_)\_$, and closure definition, $\#\_\_$, have lower precedence than other operators, and cast precedes closure definition. Hence $\#()(\mathtt{this}!())$ can be written as $\#()\mathtt{this}!()$. The, possibly indexed and/or primed, metavariables T, V, U, S, W range over type expressions; X, Y, Z range over type variables; N, P, Q range over class types; C, D, E range over class names; f, g range over field names; e, v, d range over expressions; x, y range over variable names and M, K, L and m range respectively, over methods, constructors, classes, and method names and eventually F ranges over closures.Moreover, we use $[\overline{\mathtt{e}}/\overline{\mathtt{x}}]e$ (respectively, $[\overline{\mathtt{T}}/\overline{\mathtt{X}}]T$), for value (res. type) substitution, meaning the result of the simultaneous replacing of $\overline{\mathtt{x}}$ by $\overline{\mathtt{e}}$ in e (res. $\overline{\mathtt{X}}$ by $\overline{\mathtt{T}}$ in type expression T). Eventually $FV(\overline{\mathtt{T}})$ denotes the set of type variables in $\overline{\mathtt{T}}$.

3.2. Syntax

The syntax of closures is the one of [3] and allows to define closures (*lambda expressions*) and closure types (*function types*). *Lambda expressions* consist of closures whose body is an expression and of closures whose body is a block. In FGCJ, since sequencing and assignment are omitted, the body of a closure can only be an expression. The syntax for closures is $\#(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\mathtt{e}$, where $\overline{\mathtt{x}}$ are the formal names, $\overline{\mathtt{T}}$ are the formal types and e is the body. Closure invocation operator is denoted by symbol '!', hence the syntax for closure invocation is $\mathtt{e}!(\overline{\mathtt{e}})$, where e is a closure receiving the list of arguments $\overline{\mathtt{e}}$. This syntax is motivated by the need for keeping, in Java, the method names separated from the variable names (i.e. any identifier that precedes symbol '!', is a name of a variable, possibly binding a closure) and by conciseness [23]. Other proposals [15, 10] use a more usual syntax for invocation.

Table 1

Syntax

T	::=	$\mathtt{X} \mid \mathtt{N} \mid$	$(\mathrm{T}_{\mathrm{FGJ}})$
		$\mid \#\mathtt{T}(\overline{\mathtt{T}})$	$(\mathrm{T}_{\mathrm{FGCJ}})$
N	::=	$\mathtt{C}\langle\overline{\mathtt{T}}\rangle$	$(\mathrm{N}_{\mathrm{FGJ}})$
L	::=	$\mathtt{class}\ \mathtt{C}\langle\overline{\mathtt{X}} \lhd \overline{\mathtt{N}}\rangle \lhd \mathtt{N}\{\overline{\mathtt{T}}\ \overline{\mathtt{f}}; \mathtt{K}\ \overline{\mathtt{M}}\}$	$(\mathrm{L}_{\mathrm{FGJ}})$
K	::=	$\mathtt{C}(\overline{\mathtt{T}}\ \overline{\mathtt{f}})\{\mathtt{super}(\overline{\mathtt{f}}); \mathtt{this}.\overline{\mathtt{f}} = \overline{\mathtt{f}};\}$	$(\mathrm{K}_{\mathrm{FGJ}})$
M	::=	$\langle\overline{\mathtt{X}} \lhd \overline{\mathtt{N}}\rangle\mathtt{T}\ \mathtt{m}(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\{\uparrow \mathtt{e};\}$	$(\mathrm{M}_{\mathrm{FGJ}})$
e	::=	$\mathtt{x} \mid \mathtt{e.f} \mid \mathtt{e.m}\langle\overline{\mathtt{T}}\rangle(\overline{\mathtt{e}}) \mid \mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}}) \mid (\mathtt{N})\mathtt{e} \mid$	$(\mathrm{e}_{\mathrm{FGJ}})$
		$\mid \mathtt{F} \mid \mathtt{e}\,!\,(\overline{\mathtt{e}})$	$(\mathrm{e}_{\mathrm{FGCJ}})$
F	::=	$\#(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\mathtt{e}$	$(\mathrm{F}_{\mathrm{FGCJ}})$

A closure type specifies the formal types and the result type. Hence syntax for closure types is $\#\mathtt{T}(\overline{\mathtt{T}})$. $\overline{\mathtt{T}}$ may be the empty sequence. An example is $\#(\mathtt{Integer}\ \mathtt{x}, \mathtt{Integer}\ \mathtt{y})\ (\mathtt{x} + \mathtt{y})$ which is a closure with two arguments, whose type is $\#\mathtt{Integer}(\mathtt{Integer}, \mathtt{Integer})$. No generic variables can be introduced when defining a closures but of course generic variables introduced in class or method declarations can be used inside closures. The complete syntax of the extended language is reported in `Table 1`. For the reader convenience, in all tables, but **Table 3**, the rules for FGJ have a label which is indexed by FGJ, while the rules for FGCJ have a label which is indexed by FGCJ.

3.3. Semantics: Reduction

The reduction semantics is given through the inference rules in **Table 2**, which define the reduction relation $\mathtt{e} \rightarrow \mathtt{e}'$ that says that "expression e reduces to expression $\mathtt{e}'$ in one step". The set of expressions which cannot be further reduced is the set of *normal forms* and constitute values of the calculus. In FGCJ values are not only objects but also closures, hence the following grammatical category defines the syntactic form of the values (of the value domain) of the calculus:

$$\mathtt{v} \ ::= \ \mathtt{new}\ \mathtt{N}(\overline{\mathtt{v}}) \mid \#(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\mathtt{e}$$

Hence the structure of values results from the reduction rules of the calculus. In particular in FGCJ, closure invocation needs to be considered for reduction and this is accomplished by rule GR-INVK-CLOS in **Table 2: Computation**. Such rules are those which really show how the computation is carried on. We have added rule GR-INVK-CLOS that reduces a closure invocation replacing it by the closure body in which the formal parameters are replaced by the corresponding actual ones, and `this` is replaced by the closure itself, thus allowing *recursive closures*. The rules contained in **Table 2: Congruence** are those which reduce a sub-expression contained in the expression being evaluated. We have added two rules GRC–CLOS-VAL and GRC-CLOS-ARG which consider the cases in which, the closure expression can be reduced and the case, instead, in which an actual parameter in an invoking expression can be reduced. The auxiliary functions for FGCJ are reported in **Table 3**. Actually they are the same as in [18]. We only add a case (OVER-Object) for predicate `override` when the class to which it is applied is `Object`. Examples 3.1,3.2 show how congruence rule GRC-CLOS-VAL contributes to define value structure and how it works in the reduction of recursive closures.

Example 3.1. Let $\mathtt{e} \equiv \#(\mathtt{T}\ \mathtt{x})(\#(\mathtt{T}\ \mathtt{y})\mathtt{y})!(\mathtt{x})$. Expression e is not a value since it can be further reduced: By rule GR-INV-CLOS, $(\#(\mathtt{T}\ \mathtt{y})\mathtt{y})!(\mathtt{x}) \rightarrow \mathtt{x}$, and, by rule GRC-CLOS-VAL, $\mathtt{e} \rightarrow \#(\mathtt{T}\ \mathtt{x})\mathtt{x}$. Expression $\#(\mathtt{T}\ \mathtt{x})\mathtt{x}$ is a value. On the contrary, let $\mathtt{e} \equiv \#()\mathtt{this}!()$, then expression e is a value but expression $\mathtt{e}!()$ is not a value since by rule GR-INV-CLOS, $\mathtt{e}!() \rightarrow \mathtt{e}!()$. Expression $\mathtt{e}!()$ starts an infinite sequence of reductions $e!() \rightarrow^* \mathtt{e}!()$ which is the only computation for $\mathtt{e}!()$ and is divergent. We say that $\mathtt{e}!()$ does not compute any value (i.e. it is undefined).

Example 3.2 contains an expression that yields enumerably different, finite, computations that end with the same expression which is the computed value of the expression.

Example 3.2. Let $\mathtt{T}_1 \equiv \#\mathtt{I}(\mathtt{I},\mathtt{I},\mathtt{I})$ and $\mathtt{T}_2 \equiv \#\mathtt{I}(\mathtt{I},\mathtt{I})$ be two closure types and I be a type. Let $\mathtt{e} \equiv \#(\mathtt{T}_1\mathtt{u}^?, \mathtt{T}_2\mathtt{u}^*, \mathtt{T}_2\mathtt{u}^-, \mathtt{I}\ \mathtt{u}^1, \mathtt{I}\ \mathtt{x})\mathtt{u}^?!(\mathtt{x}, \mathtt{u}^1, \mathtt{u}^*(\mathtt{x}, \mathtt{this}!(\mathtt{u}^?, \mathtt{u}^*, \mathtt{u}^-, \mathtt{u}^1, \mathtt{u}^-(\mathtt{x}, \mathtt{u}^1))))$. Expression e is a value. However, $\mathtt{e}(\mathtt{if}, *, -, 1, n) \rightarrow^* \mathtt{fact}(n)$ if I is a type for integer, `if` is a closure that computes as ordinary two-way conditional (0 is the true value), $*$ and $-$ are closures that compute as integer product

Table 2

Computation

$\dfrac{\mathit{fields}(\mathtt{N}) = \overline{\mathtt{T}}\;\overline{\mathtt{f}}}{(\mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}})).\mathtt{f}_i \longrightarrow \mathtt{e}_i}$	(GR-FIELD$_{\text{FGJ}}$)
$\dfrac{\mathit{mbody}(\mathtt{m}\langle\overline{\mathtt{V}}\rangle, \mathtt{N}) = (\overline{\mathtt{x}}, \mathtt{e})}{(\mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}})).\mathtt{m}\langle\overline{\mathtt{V}}\rangle(\overline{\mathtt{d}}) \longrightarrow [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}})/\mathtt{this}]\mathtt{e}}$	(GR-INVK$_{\text{FGJ}}$)
$\dfrac{\emptyset \vdash \mathtt{N} <: \mathtt{P}}{(\mathtt{P})(\mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}})) \rightarrow \mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}})}$	(GR-CAST$_{\text{FGJ}}$)
$\#(\overline{\mathtt{T}}\,\overline{\mathtt{x}})\mathtt{e}!(\overline{\mathtt{d}}) \longrightarrow [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \#(\overline{\mathtt{T}}\,\overline{\mathtt{x}})\mathtt{e}/\mathtt{this}]\mathtt{e}$	(GR-INV-CLOS$_{\text{FGCJ}}$)

Congruence

$\dfrac{\mathtt{e}_0 \longrightarrow \mathtt{e}_0'}{\mathtt{e}_0.\mathtt{f} \longrightarrow \mathtt{e}_0'.\mathtt{f}}$	(GRC-FIELD$_{\text{FGJ}}$)
$\dfrac{\mathtt{e}_0 \longrightarrow \mathtt{e}_0'}{\mathtt{e}_0.\mathtt{m}\langle\overline{\mathtt{T}}\rangle(\overline{\mathtt{e}}) \longrightarrow \mathtt{e}_0'.\mathtt{m}\langle\overline{\mathtt{T}}\rangle(\overline{\mathtt{e}})}$	(GRC-T-INV$_{\text{FGJ}}$)
$\dfrac{\mathtt{e}_i \longrightarrow \mathtt{e}_i'}{\mathtt{e}_0.\mathtt{m}\langle\overline{\mathtt{T}}\rangle(\ldots, \mathtt{e}_i, \ldots) \longrightarrow \mathtt{e}_0.\mathtt{m}\langle\overline{\mathtt{T}}\rangle(\ldots, \mathtt{e}_i' \ldots)}$	(GRC-INV-ARG$_{\text{FGJ}}$)
$\dfrac{\mathtt{e}_i \longrightarrow \mathtt{e}_i'}{\mathtt{new}\ \mathtt{N}(\ldots, \mathtt{e}_i, \ldots) \longrightarrow \mathtt{new}\ \mathtt{N}(\ldots, \mathtt{e}_i', \ldots)}$	(GRC-NEW$_{\text{FGJ}}$)
$\dfrac{\mathtt{e} \longrightarrow \mathtt{e}'}{(\mathtt{N})\mathtt{e} \longrightarrow (\mathtt{N})\mathtt{e}'}$	(GRC-CAST$_{\text{FGJ}}$)
$\dfrac{\mathtt{e} \longrightarrow \mathtt{e}'}{\#(\overline{\mathtt{T}}\,\overline{\mathtt{x}})\mathtt{e} \longrightarrow \#(\overline{\mathtt{T}}\,\overline{\mathtt{x}})\mathtt{e}'}$	(GRC-CLOS-VAL$_{\text{FGCJ}}$)
$\dfrac{\mathtt{e}_i \longrightarrow \mathtt{e}_i'}{\mathtt{e}!(\ldots, \mathtt{e}_i, \ldots) \longrightarrow \mathtt{e}!(\ldots, \mathtt{e}_i', \ldots)}$	(GRC-CLOS-ARG$_{\text{FGCJ}}$)

Table 3

Subclassing

$$\mathtt{C} \unlhd \mathtt{C} \qquad \frac{\mathtt{C} \unlhd \mathtt{D} \qquad \mathtt{D} \unlhd \mathtt{E}}{\mathtt{C} \unlhd \mathtt{E}} \qquad \frac{\mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{N}} \rangle \lhd \mathtt{D}\ \{\overline{\mathtt{S}}\ \overline{\mathtt{f}}; \mathtt{K}\ \overline{\mathtt{M}}\}}{\mathtt{C} \unlhd \mathtt{D}}$$

Auxiliary functions

$$\mathit{fields}(\mathtt{Object}) = \circ \qquad \text{(F-Object)}$$

$$\frac{\mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{N}} \rangle \lhd \mathtt{N}\ \{\overline{\mathtt{S}}\ \overline{\mathtt{f}}; \mathtt{K}\ \overline{\mathtt{M}}\} \qquad \mathit{fields}([\overline{\mathtt{T}}/\overline{\mathtt{X}}]\mathtt{N}) = \overline{\mathtt{U}}\ \overline{\mathtt{g}}}{\mathit{fields}(\mathtt{C}\langle \overline{\mathtt{T}} \rangle) = \overline{\mathtt{U}}\ \overline{\mathtt{g}}, [\overline{\mathtt{T}}/\overline{\mathtt{X}}]\overline{\mathtt{S}}\ \overline{\mathtt{f}}} \qquad \text{(F-Class)}$$

$$\frac{\mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{N}} \rangle \lhd \mathtt{N}\ \{\overline{\mathtt{S}}\ \overline{\mathtt{f}}; \mathtt{K}\ \overline{\mathtt{M}}\} \qquad \langle \overline{\mathtt{Y}} \lhd \overline{\mathtt{P}} \rangle \mathtt{U}\ \mathtt{m}\ (\overline{\mathtt{U}}\ \overline{\mathtt{x}})\{\uparrow \mathtt{e};\} \in \overline{\mathtt{M}}}{\mathit{mtype}(\mathtt{m}, \mathtt{C}\langle \overline{\mathtt{T}} \rangle) = [\overline{\mathtt{T}}/\overline{\mathtt{X}}](\langle \overline{\mathtt{Y}} \lhd \overline{\mathtt{P}} \rangle \overline{\mathtt{U}} \to \mathtt{U})} \qquad \text{(MT-Class)}$$

$$\frac{\mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{N}} \rangle \lhd \mathtt{N}\ \{\overline{\mathtt{S}}\ \overline{\mathtt{f}}; \mathtt{K}\ \overline{\mathtt{M}}\} \qquad \mathtt{m} \notin \overline{\mathtt{M}}}{\mathit{mtype}(\mathtt{m}, \mathtt{C}\langle \overline{\mathtt{T}} \rangle) = \mathit{mtype}(\mathtt{m}, [\overline{\mathtt{T}}/\overline{\mathtt{X}}]\mathtt{N})} \qquad \text{(MT-Super)}$$

$$\frac{\mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{N}} \rangle \lhd \mathtt{N}\ \{\overline{\mathtt{S}}\ \overline{\mathtt{f}}; \mathtt{K}\ \overline{\mathtt{M}}\} \qquad \langle \overline{\mathtt{Y}} \lhd \overline{\mathtt{P}} \rangle \mathtt{U}\ \mathtt{m}\ (\overline{\mathtt{U}}\ \overline{\mathtt{x}})\{\uparrow \mathtt{e};\} \in \overline{\mathtt{M}}}{\mathit{mbody}(\mathtt{m}\langle \overline{\mathtt{V}} \rangle, \mathtt{C}\langle \overline{\mathtt{T}} \rangle) = \overline{\mathtt{x}}.[\overline{\mathtt{T}}/\overline{\mathtt{X}}, \overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{e}} \qquad \text{(MB-Class)}$$

$$\frac{\mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{N}} \rangle \lhd \mathtt{N}\ \{\overline{\mathtt{S}}\ \overline{\mathtt{f}}; \mathtt{K}\ \overline{\mathtt{M}}\} \qquad \mathtt{m} \notin \overline{\mathtt{M}}}{\mathit{mbody}(\mathtt{m}\langle \overline{\mathtt{V}} \rangle, \mathtt{C}\langle \overline{\mathtt{T}} \rangle) = \mathit{mbody}(\mathtt{m}\langle \overline{\mathtt{V}} \rangle, [\overline{\mathtt{T}}/\overline{\mathtt{X}}]\mathtt{N})} \qquad \text{(MB-Super)}$$

Auxiliary predicates

$$\mathtt{override}(\mathtt{m}, \mathtt{Object}, \langle \overline{\mathtt{Y}} \lhd \overline{\mathtt{P}} \rangle \overline{\mathtt{T}} \to \mathtt{T}_0) \qquad \text{(Over-Object)}$$

$$\frac{\begin{array}{c}\mathit{mtype}(\mathtt{m}, \mathtt{N}) = \langle \overline{\mathtt{Z}} \lhd \overline{\mathtt{Q}} \rangle \overline{\mathtt{U}} \to \mathtt{U}_0 \implies (\overline{\mathtt{P}}, \overline{\mathtt{T}}) = [\overline{\mathtt{Y}}/\overline{\mathtt{Z}}](\overline{\mathtt{Q}}, \overline{\mathtt{U}})\ \mathtt{and} \\ \overline{\mathtt{Y}} <: \overline{\mathtt{P}} \vdash \mathtt{T}_0 <: [\overline{\mathtt{Y}}/\overline{\mathtt{Z}}]\mathtt{U}_0\end{array}}{\mathtt{override}(\mathtt{m}, \mathtt{N}, \langle \overline{\mathtt{Y}} \lhd \overline{\mathtt{P}} \rangle \overline{\mathtt{T}} \to \mathtt{T}_0)} \qquad \text{(Over)}$$

DCast

$$\frac{dcast(\mathtt{C}, \mathtt{D}) \qquad dcast(\mathtt{D}, \mathtt{E})}{dcast(\mathtt{C}, \mathtt{E})} \qquad \frac{\mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{N}} \rangle \lhd \mathtt{D}\langle \overline{\mathtt{T}} \rangle\ \{\ldots\} \quad \overline{\mathtt{X}} = FV(\overline{\mathtt{T}})}{dcast(\mathtt{C}, \mathtt{D})} \qquad \text{(DCast)}$$

and subtraction respectively and `fact` is the factorial function. In fact, $\texttt{e}(\texttt{if}, *, -, 1, n)$ has enumerably many expressions $\texttt{e}_k$ such that: $\texttt{e}(\texttt{if}, *, -, 1, n) \rightarrow^* \texttt{e}_k \rightarrow^* \texttt{fact}(n)$ (index k is the number of times expression e is replacing `this` in the reduction sequence). For instance if $n = 0$, we have:

$$\begin{array}{ll}
\texttt{e}(\texttt{if}, *, -, 1, 0) \rightarrow\ e_1 & \equiv \texttt{if!}(0, 1, *(0, \texttt{e!}(\texttt{if}, *, -, 1, -(0, 1)))) \rightarrow 1 \text{ for } k = 1 \text{ and} \\
\texttt{e}(\texttt{if}, *, -, 1, 0) \rightarrow\ e_1 \rightarrow e_2 & \equiv \texttt{if!}(0, 1, *(0, \texttt{if!}(0, 1, *(0, \texttt{e!}(\texttt{if}, *, -, 1, -(-(0, 1), 1)))))) \\
 & \quad \rightarrow 1 \text{ for } k = 2 \\
\texttt{e}(\texttt{if}, *, -, 1, 0) \rightarrow\ e_1 \rightarrow e_2 \rightarrow^* e_k \rightarrow 1 &
\end{array}$$

3.4. Semantics: Typing

The typing rules are given through inference rules that use two different kinds of environment, Δ (for type variables) and Γ (for value variables), and five different typing judgements: one for each different term structure of the language. A (well formed) type environment Δ is a finite mapping from type variables to (well formed, in Δ) types. It is written as a sequence $\texttt{X}_1$<:$\texttt{T}_1$,...,$\texttt{X}_n$<:$\texttt{T}_n$ ($\texttt{X}_i \neq \texttt{X}_j$, $i \neq j$), has domain $dom(\Delta) = \{\texttt{X}_1, ..., \texttt{X}_n\}$ and $\Delta(\texttt{X}_i) = \texttt{T}_i$ ($1 \leq i \leq n$) meaning that type variable $\texttt{X}_i$ is defined and must be bound to a subtype of type $\texttt{T}_i$. An environment Γ is defined similarly but is a finite mapping from variables to types, is written as a sequence $\texttt{x}_1\ :\ \texttt{T}_1, ..., \texttt{x}_k\ :\ \texttt{T}_k$ [1], has $dom(\Gamma) = \{\texttt{x}_1, ..., \texttt{x}_n\}$ and $\Gamma(\texttt{x}_i) = \texttt{T}_i$ ($1 \leq i \leq k$) meaning that $\texttt{x}_i$ must be bound to a value expression of type $\texttt{T}_i$.

The judgement for a (generic) type `T` (see **Table 4**) has the form $\Delta \vdash \texttt{T ok}$ meaning that `T` is a well-formed type in the (well formed) type environment Δ. The judgement for subtyping (see **Table 5**) has the form $\Delta \vdash \texttt{S}$<:`T` meaning that `S` is a subtype of `T` in Δ. The judgement for classes (see rule GT-CLASS$_{\text{FGJ}}$ in **Table 4**) has the form `C OK` meaning that `C` is well typed. The typing judgements for methods (see GT-METHOD$_{\text{FGJ}}$ in **Table 4**) has the form `M OK` in `C` meaning that `M` is well typed when its declaration occurs in class `C`. The judgement for expressions (see the first nine rules of **Table 4**) has the form $\Delta; \Gamma \vdash \texttt{e}\ :\ \texttt{T}$ meaning that expression `e` has type `T` in the typing environment Δ and in the (variable) environment Γ. The typing rules are contained in **Table 4** and extends those of FGJ. Two rules have been added for closure construction and closure invocation. Such rules simply assert the correctness of the involved types.

The rules for subtypes and wellformed types are reported in **Table 5**. Two rules are added:(WF-CLOSURE) which states that a closure type is well-formed if the types involved are well-formed and (S-CLOSURE) which states when a closure type is a subtype of another closure type, according to the contro-covariance rule for function types. Closures do not affect the bounds of type variables since in the definition of FGCJ we left unchanged the structure of the `extends` declaration of FGJ. A different situation would arise if $\langle \overline{\texttt{X}} \lhd \overline{\texttt{T}} \rangle$ replaced $\langle \overline{\texttt{X}} \lhd \overline{\texttt{N}} \rangle$ in the class and method declaration. In this case, the language would allow to express type variables that are bound to subtypes of closure types. As a consequence, the type system would require additional rules, including a rule for the $bound_\Delta$ of a closure type, to check subtyping. The rules for subtypes and wellformed types are reported in **Table 5**. Two rules are added:(WF-CLOSURE) which states that a closure type is well-formed if the types involved are well-formed and (S-CLOSURE) which states when a closure type is a subtype of another closure type, according to the contro-covariance rule for function types. Closures do not affect the bounds of type variables

[1]Variable renaming, in the program, can avoid possibly conflicts in the name of variables, without loss of generality.

Table 4

Typing rules

$$\Delta;\Gamma \vdash \mathtt{x} : \Gamma(\mathtt{x})$$
(GT-VAR$_{\text{FGJ}}$)

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

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

$$\frac{\begin{array}{c} \Delta \vdash \mathtt{N}\ \mathtt{ok} \qquad \mathit{fields}(\mathtt{N}) = \overline{\mathtt{T}}\;\overline{\mathtt{f}} \\ \Delta;\Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{S}} \qquad \Delta \vdash \overline{\mathtt{S}} <: \overline{\mathtt{T}} \end{array}}{\Delta;\Gamma \vdash \mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}}) : \mathtt{N}}$$
(GT-NEW$_{\text{FGJ}}$)

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

$$\frac{\begin{array}{c} \Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \qquad \Delta \vdash \mathtt{N}\ \mathtt{ok} \qquad \Delta \vdash \mathtt{N} <: \mathit{bound}_\Delta(\mathtt{T}_0) \\ \mathtt{N} = \mathtt{C}\langle \overline{\mathtt{T}} \rangle \qquad \mathit{bound}_\Delta(\mathtt{T}_0) = \mathtt{D}\langle \overline{\mathtt{T}} \rangle \qquad \mathit{dcast}(\mathtt{C}, \mathtt{D}) \end{array}}{\Delta;\Gamma \vdash (\mathtt{N})\mathtt{e}_0 : \mathtt{N}}$$
(GT-DCAST$_{\text{FGJ}}$)

$$\frac{\begin{array}{c} \Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \qquad \Delta \vdash \mathtt{N}\ \mathtt{ok} \\ \mathtt{N} = \mathtt{C}\langle \overline{\mathtt{T}} \rangle \qquad \mathit{bound}_\Delta(\mathtt{T}_0) = \mathtt{D}\langle \overline{\mathtt{U}} \rangle \qquad \mathtt{C} \not\trianglelefteq \mathtt{D} \qquad \mathtt{D} \not\trianglelefteq \mathtt{C} \end{array}}{\Delta;\Gamma \vdash (\mathtt{N})\mathtt{e}_0 : \mathtt{N}}$$
(GT-SCAST$_{\text{FGJ}}$)

$$\frac{\Delta \vdash \overline{\mathtt{T}}\ \mathtt{ok} \qquad \Delta;\Gamma, \overline{\mathtt{x}} : \overline{\mathtt{T}}, \mathtt{this} : \#\mathtt{T}(\overline{\mathtt{T}}) \vdash \mathtt{e} : \mathtt{T}}{\Delta;\Gamma \vdash \#(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\ \mathtt{e} : \#\mathtt{T}(\overline{\mathtt{T}})}$$
(GT-CLOSURE$_{\text{FGCJ}}$)

$$\frac{\Delta;\Gamma \vdash \mathtt{e} : \#\mathtt{T}(\overline{\mathtt{T}}) \qquad \Delta;\Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{S}} \qquad \Delta \vdash \overline{\mathtt{S}} <: \overline{\mathtt{T}}}{\Delta;\Gamma \vdash \mathtt{e!}(\overline{\mathtt{e}}) : \mathtt{T}}$$
(GT-CLOSURE-INV$_{\text{FGCJ}}$)

$$\frac{\begin{array}{c} \Delta = \overline{\mathtt{X}} <: \overline{\mathtt{N}},\ \overline{\mathtt{Y}} <: \overline{\mathtt{P}} \qquad \Delta \vdash \overline{\mathtt{T}}, \mathtt{T}, \overline{\mathtt{P}}\ \mathtt{ok} \\ \Delta; \overline{\mathtt{x}} : \overline{\mathtt{T}}, \mathtt{this} : \mathtt{C}\langle \overline{\mathtt{X}} \rangle \vdash \mathtt{e}_0 : \mathtt{S} \qquad \Delta \vdash \mathtt{S} <: \mathtt{T} \\ \mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}} \rangle \triangleleft \mathtt{N}\{\ldots\} \qquad \mathtt{override}(\mathtt{m}, \mathtt{N}, \langle \overline{\mathtt{Y}} \triangleleft \overline{\mathtt{P}} \rangle \overline{\mathtt{T}} \to \mathtt{T}) \end{array}}{\langle \overline{\mathtt{Y}} \triangleleft \overline{\mathtt{P}} \rangle \mathtt{T}\ \mathtt{m}(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\{\uparrow \mathtt{e}_0;\}\ \mathtt{OK}\ \mathtt{IN}\ \mathtt{C}\langle \overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}} \rangle}$$
(GT-METHOD$_{\text{FGJ}}$)

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

Table 5

Subtypes

$$bound_{\Delta}(\mathtt{X}) = \Delta(\mathtt{X}) \qquad (\text{B-VAR}_{\text{FGJ}})$$

$$bound_{\Delta}(\mathtt{N}) = \mathtt{N} \qquad (\text{B-CLASS}_{\text{FGJ}})$$

$$\Delta \vdash \mathtt{T} <: \mathtt{T} \qquad (\text{S-REFL}_{\text{FGJ}})$$

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

$$\Delta \vdash \mathtt{X} <: \Delta(\mathtt{X}) \qquad (\text{S-VAR}_{\text{FGJ}})$$

$$\frac{\mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{N}} \rangle \lhd \mathtt{N}\{\ldots\}}{\Delta \vdash \mathtt{C}\langle \overline{\mathtt{T}} \rangle <: [\overline{\mathtt{T}}/\overline{\mathtt{X}}]\mathtt{N}} \qquad (\text{S-CLASS}_{\text{FGJ}})$$

$$\frac{\mathtt{T} <: \mathtt{S} \qquad \overline{\mathtt{S}} <: \overline{\mathtt{T}}}{\Delta \vdash \#\mathtt{T}(\overline{\mathtt{T}}) <: \#\mathtt{S}(\overline{\mathtt{S}})} \qquad (\text{S-CLOSURE}_{\text{FGCJ}})$$

Well-formed types

$$\Delta \vdash \text{Object}\ \mathtt{ok} \qquad (\text{WF-OBJECT}_{\text{FGJ}})$$

$$\frac{\mathtt{X} \in dom(\Delta)}{\Delta \vdash \mathtt{X}\ \mathtt{ok}} \qquad (\text{WF-VAR}_{\text{FGJ}})$$

$$\frac{\mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{N}} \rangle \lhd \mathtt{N}\{\ldots\} \qquad \Delta \vdash \overline{\mathtt{T}}\ \mathtt{ok} \qquad \Delta \vdash \overline{\mathtt{T}} <: [\overline{\mathtt{T}}/\overline{\mathtt{X}}]\overline{\mathtt{N}}}{\Delta \vdash \mathtt{C}\langle \overline{\mathtt{T}} \rangle\ \mathtt{ok}} \qquad (\text{WF-CLASS}_{\text{FGJ}})$$

$$\frac{\Delta \vdash \overline{\mathtt{T}}\ \mathtt{ok} \qquad \Delta \vdash \mathtt{T}\ \mathtt{ok}}{\Delta \vdash \#\mathtt{T}(\overline{\mathtt{T}})\ \mathtt{ok}} \qquad (\text{WF-CLOSURE}_{\text{FGCJ}})$$

since in the definition of FGCJ we left unchanged the structure of the `extends` declaration of FGJ. A different situation would arise if $\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{T}} \rangle$ replaced $\langle \overline{\mathtt{X}} \lhd \overline{\mathtt{N}} \rangle$ in the class and method declaration. In this case, the language would allow to express type variables that are bound to subtypes of closure types. As a consequence, the type system would require additional rules, including a rule for the $bound_{\Delta}$ of a closure type, to check subtyping.

Example 3.3. Let $\Delta \equiv \Delta_1, \mathtt{T}_1{<:}\mathtt{A}, \mathtt{T}_2{<:}\mathtt{B}, \mathtt{T}_3{<:}\mathtt{C}, \mathtt{T}_4{<:}\mathtt{D}$, and $\Delta \vdash \mathtt{C}{<:}\mathtt{A}$ $\Delta \vdash \mathtt{B}{<:}\mathtt{D}$ for classes `A`, `B`, `C`, `D`. Then $\Delta \vdash \#\mathtt{T}_1(\mathtt{T}_2)$ `OK` and $\Delta \vdash \#\mathtt{T}_3(\mathtt{T}_4)$ `OK`, but $\#\mathtt{T}_1(\mathtt{T}_2)$, $\#\mathtt{T}_3(\mathtt{T}_4)$ are uncomparable types under Δ. Any attempt to use an expression of type $\#\mathtt{T}_3(\mathtt{T}_4)$ instead of an expression of type $\#\mathtt{T}_1(\mathtt{T}_2)$ yields a type error, as in the statement: `new N(e)`, where `e` is an expression of type $\#\mathtt{T}_3(\mathtt{T}_4)$ and $field(\mathtt{N}) \equiv \#\mathtt{T}_1(\mathtt{T}_2)$ `f`. On the contrary, each of the following type assignments are correct for `new N(e)`: $field(\mathtt{N}) \equiv \#\mathtt{A}(\mathtt{T}_2)$ `f`, and `e`$:\#\mathtt{T}_1(\mathtt{T}_2)$; $field(\mathtt{N}) \equiv \#\mathtt{A}(\mathtt{T}_2)$ `f`, and `e`$:\#\mathtt{T}_1(\mathtt{B})$; $field(\mathtt{N}) \equiv \#\mathtt{D}(\mathtt{C})$ `f`, and `e`$:\#\mathtt{B}(\mathtt{T}_3)$.

4. Properties

Semantics is useful to prove language properties. In this paper we prove type soundness which states that an expression and its normal form have compatible types, see Section 4.1. Successively we prove backward compatibility which states that programs, in the kernel language, maintain their meaning, in the extended language. Eventually, we prove the abstraction property which states that a closure acts as a code abstraction.

4.1. Type Soundness

Analogously to [18] we prove subject reduction theorem and progress theorem, type soundness immediately follows. Several interesting lemmas are used in the proofs. They are stated and proved in the appendix.

Theorem 4.1. (Subject reduction)
If $\Delta;\Gamma \vdash \mathtt{e} : \mathtt{T}$ and $\mathtt{e} \rightarrow \mathtt{e}'$ then $\Delta;\Gamma \vdash \mathtt{e}' : \mathtt{T}'$, for some $\mathtt{T}'$ such that $\Delta \vdash \mathtt{T}'{<:}\mathtt{T}$

Proof: See Appendix A □

Theorem 4.2. (Progress)
Suppose `e` is a well-typed expression. If `e` includes as a subexpression:

1. $\mathtt{new\ N}(\overline{\mathtt{e}}).\mathtt{f}$ then $fields(\mathtt{N}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}$, for some $\overline{\mathtt{T}}$ and $\overline{\mathtt{f}}$, and $\mathtt{f} \in \overline{\mathtt{f}}$.

2. $\mathtt{new\ N}(\overline{\mathtt{e}}).\mathtt{m}\langle\overline{\mathtt{V}}\rangle(\overline{\mathtt{d}})$ then $mbody(\mathtt{m}\langle\overline{\mathtt{V}}\rangle, \mathtt{N}) = \overline{\mathtt{x}}.\mathtt{e}_0$, for some $\overline{\mathtt{x}}$ and $\mathtt{e}_0$, and $|\overline{\mathtt{x}}| = |\overline{\mathtt{d}}|$.

3. $\mathtt{F}!(\overline{\mathtt{d}})$ then $\mathtt{F} = \#(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\ \mathtt{e}_0$, for some $\overline{\mathtt{T}}$, $\overline{\mathtt{x}}$ and $\mathtt{e}_0$, and $|\overline{\mathtt{x}}| = |\overline{\mathtt{d}}|$.

Proof: The proof is based on the analysis of all well typed expressions, which can be reduced to the above 3 cases to conclude that either it is in normal form or it can be further reduced to obtain a normal form. As already stated in section 3.3, in FGCJ there are 2 possible normal forms i.e. values. They are: $\mathtt{new\ N}(\overline{\mathtt{w}})$ (Object in FGJ), and $\#(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\mathtt{e}$ (closure) □

Theorem 4.3. (Type Soundness)
If $\emptyset;\emptyset \vdash \mathtt{e} : \mathtt{T}$ and $\mathtt{e} \rightarrow^* \mathtt{e}'$ with $\mathtt{e}'$ a normal form, then $\mathtt{e}'$ is either (1) a value `w` with $\emptyset;\emptyset \vdash \mathtt{w} : \mathtt{S}$ and $\emptyset \vdash \mathtt{S}{<:}\mathtt{T}$ or (2) an expression containing $(\mathtt{P})\ \mathtt{new\ N}(\overline{\mathtt{e}})$ where $\mathtt{N} \not<: \mathtt{P}$

Proof: Immediate from above Theorems □

4.2. Computation properties

We prove that the extension made to the language, to add closures, preserves the meaning of the programs of the original language FGJ.

Theorem 4.4. (Backward compatibility)
If an FGJ program is well typed under the FGJ rules it is also well typed under the FGCJ rules. Moreover, for all FGJ programs e and e' (whether well typed or not) $\mathtt{e} \rightarrow_{\mathrm{FGJ}} \mathtt{e}' \iff \mathtt{e} \rightarrow_{\mathrm{FGCJ}} \mathtt{e}'$.

Proof: All FGCJ sets of rules include FGJ rules □

4.3. Abstraction Property

This property concerns the use of simple closures in code abstractions and is the analog in FGCJ of $\beta-$conversion in Lambda Calculus. $\beta-$conversion states that $(\lambda\mathtt{x}.\mathtt{e}[\mathtt{x}])\mathtt{e}' = \mathtt{e}[\mathtt{e}']$ for all Lambda expressions $\mathtt{e}[\mathtt{e}']$, $\mathtt{e}'$ (possibly, after variable renaming to avoid name collisions), and a fresh variable[2] x [1]. Unfortunately, this property cannot hold for closures in FGCJ: A counterexample is the pair of expressions e and u of example 4.1. In fact, in addition to variables, the expressions of FGCJ contain four kinds of identifiers for naming classes, fields, methods, and a special identifier `this` for self-reference.

Definition 4.1. (context e[•], type and substitution)
The set of *contexts* $\mathtt{e}[\bullet]$ is:

$$\begin{array}{lcl} \mathtt{e}[\bullet] & ::= & \bullet \mid \mathtt{x} \mid \mathtt{e}[\bullet].\mathtt{f} \mid \mathtt{e}[\bullet].\mathtt{m}\langle\overline{\mathtt{T}}\rangle(\overline{\mathtt{e}[\bullet]}) \mid \mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}[\bullet]}) \mid (\mathtt{N})\mathtt{e}[\bullet] \\ & & \mid \mathtt{F}[\bullet] \mid \mathtt{e}[\bullet]\,!\,(\overline{\mathtt{e}[\bullet]}) \\ \mathtt{F}[\bullet] & ::= & \#(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\mathtt{e}[\bullet] \end{array}$$

The *substitution* of expression $\mathtt{e}_1$ in the context $\mathtt{e}[\bullet]$ is the expression $\mathtt{e}[\mathtt{e}_1]$ obtained replacing $\mathtt{e}_1$ in each hole of $\mathtt{e}[\bullet]$.
A *context of type* $(\Gamma, \mathtt{T})$ is any context $\mathtt{H}[\bullet]$, in FGCJ, such that $\Delta; \Gamma, \mathtt{x} : \mathtt{T} \vdash \mathtt{H}[\mathtt{x}] : \mathtt{S}$ for some $\Delta \vdash \mathtt{T}\ \mathtt{ok}$ and type S, and fresh variable x.

Free and Bound Identifiers. Contexts, as well as expressions, may contain free variables, field and method identifiers, and occurrences of `this` as it is when they are the result of a textual extraction of expressions from method (or closure) bodies. In FGCJ, variables are declared only in the parameter list of either methods or closures. Hence, the free variables of a context, or expression, are all the variables that do not occur inside a closure (contained in the context, or expression) and all those that occur inside a closure but not in its parameter list. Class identifiers are assumed to be unique in each program. Hence, they are always bounds (to the corresponding class in the program class table, CT [18]). Moreover, field (and method) identifiers may occur only in terms of the form (i) $((\mathtt{N})\mathtt{e}).\mathtt{f}$ or (ii) $\mathtt{e}_0.\cdots\mathtt{e}_n.\mathtt{f}$, where f is a field identifier, $(\mathtt{N})\mathtt{e}$ and $\mathtt{e}_0.\cdots\mathtt{e}_n$ express the target object. Hence, they are always bound (to the corresponding field, method, of the class N in case (i), of the class type of $\mathtt{e}_0.\cdots\mathtt{e}_n$ in case (ii)). On the contrary, the self reference `this` occurs bound, in a context (or an expression), only when it occurs inside a closure defined in such a context (or expression). In all the other cases, `this` occurs free.

[2]a variable that does not occur in the expression $\mathtt{e}[\mathtt{e}']$

Definition 4.2. (equivalence: $\approx$)
Two expressions $\mathtt{e}_1$ and $\mathtt{e}_2$ are equivalent if they compute in the same way[3], i.e.: expression $\mathtt{e}'$ exists such that $\mathtt{e}_1 \rightarrow^* \mathtt{e}'$ and $\mathtt{e}_2 \rightarrow^* \mathtt{e}'$

Theorem 4.5. (Abstraction Property)
Let $\Delta \vdash \mathtt{T}\ \mathtt{ok}$, $\mathtt{H}[\bullet]$ be any context, $\mathtt{G}[\bullet]$ be any context of type $(\Gamma, \mathtt{T})$ and with no free occurrences of `this`. Let $\mathtt{e}_2$ be any expression such that its free variables[4] and free occurrences of `this` are not bound in $\mathtt{e}_1 \equiv \mathtt{G}[\mathtt{e}_2]$ (but possibly, in $\mathtt{H}[\bullet]$). Then $\mathtt{H}[(\#(\mathtt{T}\ \mathtt{x})\mathtt{G}[\mathtt{x}])!(\mathtt{e}_2)] \approx \mathtt{H}[\mathtt{e}_1]$, for any fresh variable $\mathtt{x}$.

Proof: See Appendix A □

Example 4.1. Consider expression $\mathtt{e} \equiv \mathtt{new\ Pair}\langle\mathtt{Z},\mathtt{Y}\rangle(\mathtt{newfst}, \mathtt{this.snd})$ which is the body of `setfst`, defined in Section 2. Expressions `u`, `v` and `w` are three different rearrangements of `e` which generalize, through closures, different subterms of `e`

$\mathtt{u} \equiv (\#(\mathtt{Z}\ \mathtt{x})\ \mathtt{new\ Pair}\langle\mathtt{Z},\mathtt{Y}\rangle(\mathtt{x}, \mathtt{this.snd}))\,!\,(\mathtt{newfst})$

$\mathtt{v} \equiv (\#(\mathtt{Y}\ \mathtt{y})\ \mathtt{new\ Pair}\langle\mathtt{Z},\mathtt{Y}\rangle(\mathtt{newfst},\mathtt{y}))\,!\,(\mathtt{this.snd})$

$\mathtt{w} \equiv (\#(\mathtt{Y}\ \mathtt{y})(\#(\mathtt{Z}\ \mathtt{x})\ \mathtt{new\ Pair}\langle\mathtt{Z},\mathtt{Y}\rangle(\mathtt{x}, \mathtt{y}))\,!\,(\mathtt{newfst}))\,!\,(\mathtt{this.snd})$

According to Theorem 4.5, $\mathtt{e} \approx \mathtt{v}$, and $\mathtt{v} \approx \mathtt{w}$ but $\mathtt{e} \not\approx \mathtt{u}$ (since for `u`: $\mathtt{H}[\bullet] \equiv \bullet$ and $\mathtt{G}[\bullet] \equiv \mathtt{new\ Pair}\langle\mathtt{Z},\mathtt{Y}\rangle(\bullet, \mathtt{this.snd})$ which contains a free occurrence of `this`). Hence, we can replace `e` with either `v` or `w`, but the replacement of `e` with `u` in the body of method `setfst` yields a different meaning of the program.

5. Conclusion

In this paper we address the problem of defining a formal semantics to prove properties of closures in Java, according to the Straw-man proposal. A minimal calculus is defined extending Featherweight Java, both for syntax and semantics and three properties: type soundness, backward compatibility and the abstraction property are proved. In [7] a translation semantics for a somewhat different proposal for closures in Java is defined. Such translational approach defines a rigorous semantics for closures and provides also an implementation for Java extended with closures, but is not suited to prove semantic properties. We plan to adapt translation semantics to Simple Closures and analogously to [18] prove that the two semantics commute. Eventually this paper is a partial result of a more complex and ambitious project of extension and prototype implementation of Java with higher order features, besides closures, methods passed as parameters [6].

[3]It includes the case in which both expressions diverge, but it excludes the case in which one or both are illegal, wrong, untypeable terms.
[4]The requirement on free variables could be omitted resorting to variable renaming.

A. Lemmas and Theorem Proofs

Lemmas A.2.1. through A.2.9. and A.2.12 in [18] remain valid for FGCJ without proof extensions and are not reported here. Proofs of Lemma A.2.10 and A.2.11 need to be extended and are reported below.

Lemma A.2.10. *If* $\Delta_1, \overline{\mathtt{X}}{<:}\overline{\mathtt{N}}, \Delta_2; \Gamma \vdash \mathtt{e} : \mathtt{T}$ *and* $\Delta_1 \vdash \overline{\mathtt{U}}{<:}[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{N}}$ *where* $\Delta_1 \vdash \overline{\mathtt{U}}\ \mathtt{ok}$ *and none of* $\overline{\mathtt{X}}$ *appears in* Δ_1*, then* $\Delta_1, [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\Delta_2; [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\Gamma \vdash [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{e} : \mathtt{S}$ *for some* $\mathtt{S}$ *such that* $\Delta_1, [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\Delta_2 \vdash \mathtt{S}{<:}[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{T}$

Proof: As in [18], the proof is given by induction and case analysis. We specify only the new cases.

Case GT-CLOSURE

$$\mathtt{e} = \#(\overline{\mathtt{W}}\,\overline{\mathtt{w}})\ \mathtt{e}_0 \quad \Delta = \Delta_1,\ \overline{\mathtt{X}}{<:}\overline{\mathtt{N}}, \Delta_2$$
$$\Delta; \Gamma \vdash \mathtt{e}_0 : \mathtt{W} \quad \mathtt{T} = \#\mathtt{W}(\overline{\mathtt{W}})$$
$$[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{e} = \#([\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{W}}\,\overline{\mathtt{w}})[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{e}_0$$

Hence $\mathtt{S} = \#[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{W}([\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{W}}) = [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{T}$.

Case GT-CLOSURE-INV

$$\mathtt{e} = \mathtt{e}_0\,\mathtt{!}\,(\overline{\mathtt{e}}) \quad \mathtt{T} = \mathtt{W}$$
$$\Delta; \Gamma \vdash \mathtt{e}_0 : \#\mathtt{W}(\overline{\mathtt{W}}) \quad \Delta; \Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{S}}{<:}\overline{\mathtt{W}}$$
$$[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{e} = [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{e}_0\,\mathtt{!}\,([\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{e}})$$

By induction hypothesis $\Delta; \Gamma \vdash [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{e}} : \overline{\mathtt{Q}}$ such that $\Delta \vdash \overline{\mathtt{Q}}{<:}[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{S}}$. By Lemma A.2.5, since $\Delta \vdash \overline{\mathtt{S}}{<:}\overline{\mathtt{W}}$ then $\Delta \vdash [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{S}}{<:}[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{W}}$ and $\Delta \vdash \overline{\mathtt{Q}}{<:}[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{W}}$ by rule S-TRANS. Furthemore, by induction hypothesis $\Delta; \Gamma \vdash [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{e}_0 : \mathtt{Q}$ such that $\Delta \vdash \mathtt{Q}{<:}[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\#(\mathtt{W}(\overline{\mathtt{W}})) = \#[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{W}([\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{W}})$. By Lemma Closure Subtyping, $\mathtt{Q} \equiv \#\mathtt{V}(\overline{\mathtt{V}})$ and $\mathtt{V}{<:}[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{W}$ and $[\overline{\mathtt{U}}/\overline{\mathtt{X}}](\overline{\mathtt{W}})){<:}\overline{\mathtt{V}}$, and by rule S-TRANS $[\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{S}}{<:}\overline{\mathtt{V}}$. Hence, by rule GT-CLOSURE-INV $\Delta; \Gamma \vdash [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{e}_0\,\mathtt{!}\,([\overline{\mathtt{U}}/\overline{\mathtt{X}}]\overline{\mathtt{e}}) : [\overline{\mathtt{U}}/\overline{\mathtt{X}}]\mathtt{W}$ □

Lemma A.2.11. *If* $\Delta; \Gamma, \overline{\mathtt{x}} : \overline{\mathtt{T}} \vdash \mathtt{e} : \mathtt{T}$ *and* $\Delta; \Gamma \vdash \overline{\mathtt{d}} : \overline{\mathtt{S}}$ *where* $\Delta \vdash \overline{\mathtt{S}}{<:}\overline{\mathtt{T}}$*, then* $\Delta; \Gamma \vdash [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e} : \mathtt{S}$ *for some* $\mathtt{S}$ *such that* $\Delta \vdash \mathtt{S}{<:}\mathtt{T}$

Proof: As in [18], the proof is given by induction and case analysis. We specify only the new cases.

Case GT-CLOSURE

$$\mathtt{e} = \#(\overline{\mathtt{W}}\,\overline{\mathtt{y}})\mathtt{e}_0 \quad \Delta \vdash \overline{\mathtt{W}}, \mathtt{W}\ \mathtt{ok}$$
$$\Delta; \Gamma, \overline{\mathtt{y}} : \overline{\mathtt{W}} \vdash \mathtt{e}_0 : \mathtt{W}_0 \quad \Delta \vdash \mathtt{W}_0{<:}\mathtt{W} \quad \mathtt{T} = \#\mathtt{W}(\overline{\mathtt{W}})$$

By the induction hypothesis, $\Delta; \Gamma \vdash [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0 : \mathtt{W}_1$ for some $\mathtt{W}_1$ such that $\Delta \vdash \mathtt{W}_1{<:}\mathtt{W}_0$ and $[\overline{\mathtt{d}}/\overline{\mathtt{x}}](\#(\overline{\mathtt{W}}\,\overline{\mathtt{y}})\mathtt{e}_0) = \#(\overline{\mathtt{W}}\,\overline{\mathtt{y}})[\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0$, because of renaming to avoid name coalashing, $\overline{\mathtt{x}} \cap \overline{\mathtt{y}} = \{\}$. Hence GT-CLOSURE premises are satisfied and $\mathtt{S} = \#\mathtt{W}_1(\overline{\mathtt{W}}){<:}\#\mathtt{W}(\overline{\mathtt{W}})$.

Case GT-CLOSURE-INV

$$\mathtt{e} = \mathtt{e}_0\,\mathtt{!}\,(\overline{\mathtt{e}}) \quad \Delta; \Gamma \vdash \mathtt{e}_0 : \#\mathtt{W}(\overline{\mathtt{W}})$$
$$\Delta; \Gamma, \overline{\mathtt{e}} : \overline{\mathtt{S}} \quad \Delta \vdash \overline{\mathtt{S}}{<:}\overline{\mathtt{W}} \quad \mathtt{T} = \mathtt{W}$$

By the induction hypothesis, $\Delta; \Gamma \vdash [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0 : \mathtt{W}_0$ for some $\mathtt{W}_0$ such that $\Delta \vdash \mathtt{W}_0{<:}\#\mathtt{W}(\overline{\mathtt{W}})$ and $\Delta; \Gamma \vdash [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\overline{\mathtt{e}} : \overline{\mathtt{Q}}$ for some $\overline{\mathtt{Q}}{<:}\overline{\mathtt{W}}$ because of subtyping on closures $\mathtt{W}_0 = \#\mathtt{P}(\overline{\mathtt{P}}){<:}\#\mathtt{W}(\overline{\mathtt{W}})$ and $\mathtt{P}{<:}\mathtt{W}$ and $\overline{\mathtt{W}}{<:}\overline{\mathtt{P}}$ and for transitivity $\overline{\mathtt{Q}}{<:}\overline{\mathtt{P}}$ hence, by rule GT-CLOSURE-INV, $\mathtt{S} = \mathtt{P}$ □

Lemma Closure Subtyping. *If* $\Delta \vdash \mathtt{W}, \mathtt{T}, \overline{\mathtt{T}}\ \mathtt{ok}$ *then* $\Delta \vdash \mathtt{W}{<:}\mathtt{T}(\overline{\mathtt{T}}) \iff \mathtt{W} \equiv \mathtt{S}(\overline{\mathtt{S}})\ and\ \mathtt{S}{<:}\mathtt{T}\ and\ \overline{\mathtt{T}}{<:}\overline{\mathtt{S}}$

Proof: Immediate after subtyping rules □

Proof of Theorem 4.1 By induction on the reduction $\mathtt{e} \to \mathtt{e}'$, with a case analysis on the reduction rule used. It extends the proof, [18] (pp. 426-428), of the corresponding theorem for FGJ with the following cases;

Case GR-INV-CLOS $\mathtt{e} = \#(\overline{\mathtt{T}}\,\overline{\mathtt{x}})\mathtt{e}_0\,!\,(\overline{\mathtt{d}})$ and by rule GT-CLOSURE we have: $\Delta;\Gamma \vdash \#(\overline{\mathtt{T}}\,\overline{\mathtt{x}})\mathtt{e}_0 : \#\mathtt{T}(\overline{\mathtt{T}})$ and $\Delta;\Gamma,\overline{\mathtt{x}} : \overline{\mathtt{T}}, \mathtt{this} : \#\mathtt{T}(\overline{\mathtt{T}}) \vdash \mathtt{e}_0 : \mathtt{T}$ and by rule GT-CLOSURE-INV, $\Delta;\Gamma \vdash \overline{\mathtt{d}} : \overline{\mathtt{S}}$, $\overline{\mathtt{S}}{<:}\overline{\mathtt{T}}$ and $\Delta;\Gamma \vdash \mathtt{e} : \mathtt{T}$. Furthermore $\mathtt{e}' = [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \#(\overline{\mathtt{T}}\,\overline{\mathtt{x}})\mathtt{e}_0/\mathtt{this}]\mathtt{e}_0$ and by the Lemma A.2.11 $\Delta;\Gamma \vdash \mathtt{e}' = [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \#(\overline{\mathtt{T}}\,\overline{\mathtt{x}})\mathtt{e}_0/\mathtt{this}]\mathtt{e}_0 : \mathtt{R}$ for some $\mathtt{R}{<:}\mathtt{T}$.

Case GRC-CLOS-VAL and GRC-CLOS-ARG Easy □

Lemma β-conversion *Let* $\Delta \vdash \mathtt{T}\ \mathtt{ok}$, $\mathtt{H}[\bullet]$ *be any context of type* $(\Gamma_1, \mathtt{T})$ *and with no free occurrences of* $\mathtt{this}$. *Let* $\mathtt{e}_1$ *be an expression of type* $\mathtt{T}$ *(for* $\mathtt{H}[\bullet]$, *i.e.* $\Delta;\Gamma_1,\Gamma_2 \vdash \mathtt{e}_1 : \mathtt{T}$*) such that all the free variables of* $\mathtt{e}_1$ *are not bound in* $\mathtt{H}[\bullet]$. *Then* $(\#(\mathtt{T}\ \mathtt{x})\mathtt{H}[\mathtt{x}])!(\mathtt{e}_1) \approx \mathtt{H}[\mathtt{e}_1]$ *for any fresh variable* $\mathtt{x}$.

Proof: By case analysis on the structure of context $\mathtt{H}[\bullet]$.

Case $\mathtt{H}[\bullet] \equiv \bullet$. Then, $(\#(\mathtt{T}\,\mathtt{x})\mathtt{H}[\mathtt{x}])!(\mathtt{e}_1) = (\#(\mathtt{T}\,\mathtt{x})\mathtt{x})!(\mathtt{e}_1)$ and $\mathtt{H}[\mathtt{e}_1] = \mathtt{e}_1$. Moreover, by rule GR-INVK-CLOS$_{\text{FGCJ}}$, $(\#(\mathtt{T}\,\mathtt{x})\mathtt{x})!(\mathtt{e}_1) \to [\mathtt{e}_1/\mathtt{x}, (\#(\mathtt{T}\,\mathtt{x})\mathtt{x})/\mathtt{this}]\mathtt{x}$. Hence,
$[\mathtt{e}_1/\mathtt{x}, (\#(\mathtt{T}\,\mathtt{x})\mathtt{x})/\mathtt{this}]\mathtt{x} \equiv \mathtt{e}_1$.

Case $\mathtt{H}[\bullet] \equiv \mathtt{y}$, where $\mathtt{y} \neq \mathtt{this}$ and $\mathtt{y} \neq \mathtt{x}$. Then, $(\#(\mathtt{T}\,\mathtt{x})\mathtt{H}[\mathtt{x}])!(\mathtt{e}_1) = (\#(\mathtt{T}\,\mathtt{x})\mathtt{y})!(\mathtt{e}_1)$ and $\mathtt{H}[\mathtt{e}_1] = \mathtt{y}$. Moreover, by rule GR-INVK-CLOS$_{\text{FGCJ}}$,
$(\#(\mathtt{T}\,\mathtt{x})\mathtt{y})!(\mathtt{e}_1) \to [\mathtt{e}_1/\mathtt{x}, (\#(\mathtt{T}\,\mathtt{x})\mathtt{x})/\mathtt{this}]\mathtt{y}$. Hence, $[\mathtt{e}_1/\mathtt{x}, (\#(\mathtt{T}\,\mathtt{x})\mathtt{x})/\mathtt{this}]\mathtt{y} \equiv \mathtt{y}$, since assumptions on $\mathtt{y}$.

Case $\mathtt{H}[\bullet] \equiv \mathtt{e}[\bullet].\mathtt{f}$ where $\mathtt{this}$ does not occur free in $\mathtt{e}[\bullet]$. Then, $(\#(\mathtt{T}\,\mathtt{x})\mathtt{H}[\mathtt{x}])!(\mathtt{e}_1) =$
$(\#(\mathtt{T}\,\mathtt{x})\mathtt{e}[\mathtt{x}].\mathtt{f})!(\mathtt{e}_1)$ and $\mathtt{H}[\mathtt{e}_1] = \mathtt{e}[\mathtt{e}_1].\mathtt{f}$. Moreover, by rule GR-INVK-CLOS$_{\text{FGCJ}}$,
$(\#(\mathtt{T}\,\mathtt{x})\mathtt{e}[\mathtt{x}].\mathtt{f})!(\mathtt{e}_1) \to [\mathtt{e}_1/\mathtt{x}, \#(\mathtt{T}\,\mathtt{x})\mathtt{x}/\mathtt{this}]\mathtt{e}[\mathtt{x}].\mathtt{f}$. By def. of substitution,
$[\mathtt{e}_1/\mathtt{x}, \#(\mathtt{T}\,\mathtt{x})\mathtt{x}/\mathtt{this}]\mathtt{e}[\mathtt{x}].\mathtt{f} = ([\mathtt{e}_1/\mathtt{x}, \#(\mathtt{T}\,\mathtt{x})\mathtt{x}/\mathtt{this}]\mathtt{e}[\mathtt{x}]).([\mathtt{e}_1/\mathtt{x}, \#(\mathtt{T}\,\mathtt{x})\mathtt{x}/\mathtt{this}]\mathtt{f})$.
By assumption on the free occurrences of $\mathtt{this}$,
$([\mathtt{e}_1/\mathtt{x}, \#(\mathtt{T}\,\mathtt{x})\mathtt{x}/\mathtt{this}]\mathtt{e}[\mathtt{x}]).([\mathtt{e}_1/\mathtt{x}, \#(\mathtt{T}\,\mathtt{x})\mathtt{x}/\mathtt{this}]\mathtt{f}) = ([\mathtt{e}_1/\mathtt{x}]\mathtt{e}[\mathtt{x}]).([\mathtt{e}_1/\mathtt{x}]\mathtt{f}) = \mathtt{e}[\mathtt{e}_1].\mathtt{f}$

Case $\mathtt{H}[\bullet] \equiv \mathtt{e}[\bullet].\mathtt{m}\langle\overline{\mathtt{T}}\rangle(\overline{\mathtt{e}[\bullet]})$, $\mathtt{H}[\bullet] \equiv \mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}[\bullet]})$, $\mathtt{H}[\bullet] \equiv (\mathtt{N})\mathtt{e}[\bullet], \mathtt{H}[\bullet] \equiv \mathtt{F}[\bullet]$, $\mathtt{H}[\bullet] \equiv \mathtt{e}[\bullet]\,!\,(\overline{\mathtt{e}[\bullet]})$ can be similarly proved □

Proof of Theorem 4.5 By induction on the structure of contexts.

Case $\mathtt{H}[\bullet] \equiv \bullet$. Then $\mathtt{H}[(\#(\mathtt{T}\,\mathtt{x})\mathtt{G}[\mathtt{x}])!(\mathtt{e}_2)] = (\#(\mathtt{T}\,\mathtt{x})\mathtt{G}[\mathtt{x}])!(\mathtt{e}_2)$ and $\mathtt{H}[\mathtt{e}_1] = \mathtt{e}_1$, and, by Lemma β-conversion, the case holds.

Case $\mathtt{H}[\bullet] \equiv \mathtt{y}$. Then $\mathtt{H}[(\#(\mathtt{T}\,\mathtt{x})\mathtt{G}[\mathtt{x}])!(\mathtt{e}_2)] = \mathtt{y}$ and $\mathtt{H}[\mathtt{e}_1] = \mathtt{y}$ and the case holds.

Case $\mathtt{H}[\bullet] \equiv \mathtt{e}[\bullet]\mathtt{.f}$. Assume that theorem holds for context $\mathtt{e}[\bullet]$: (ih) $\mathtt{e}[(\#(\mathtt{T\,x})\mathtt{G}[\mathtt{x}])!(\mathtt{e}_2)] \approx \mathtt{e}[\mathtt{e}_1]$ (i.e: $\mathtt{e}[(\#(\mathtt{T\,x})\mathtt{G}[\mathtt{x}])!(\mathtt{e}_2)] \rightarrow^* \mathtt{e}'$ and $\mathtt{e}[\mathtt{e}_1] \rightarrow^* \mathtt{e}'$ for some $\mathtt{e}'$). Then, $\mathtt{H}[(\#(\mathtt{T\,x})\mathtt{G}[\mathtt{x}])!(\mathtt{e}_2)] = \mathtt{e}[(\#(\mathtt{T\,x})\mathtt{G}[\mathtt{x}])!(\mathtt{e}_2)]\mathtt{.f}$ and $\mathtt{H}[\mathtt{e}_1] = \mathtt{e}[\mathtt{e}_1]\mathtt{.f}$, and by GRC-FIELD$_{\text{FGJ}}$ the case holds.

Case $\mathtt{H}[\bullet] \equiv \mathtt{e}[\bullet]\mathtt{.m}\langle\overline{\mathtt{T}}\rangle(\overline{\mathtt{e}[\bullet]})$, $\mathtt{H}[\bullet] \equiv \mathtt{new\ N}(\overline{\mathtt{e}[\bullet]})$, $\mathtt{H}[\bullet] \equiv (\mathtt{N})\mathtt{e}[\bullet], \mathtt{H}[\bullet] \equiv \mathtt{F}[\bullet]$, $\mathtt{H}[\bullet] \equiv \mathtt{e}[\bullet]\mathtt{!}(\overline{\mathtt{e}[\bullet]})$ can be similarly proved □

References

[1] H.P. Barendregt. Functional Pgramming and Lambda Calculus. *Handbook Of Theoretical Computer Science*, 2:321–363, 1990.

[2] G. Bracha, N. Gafter, J. Gosling, and P. von der Ahe. Closures for Java, 2006. http://blogs.sun.com/ahe/resource/closures.pdf.

[3] A. Buckley, J. Gibbons, and M. Reinhold. *Project Lambda: Java Language Specification. Version 0.1*. Sun Microsystem, Inc., January 2010. http://mail.openjdk.java.net/pipermail/lambda-dev/attachments/20100122/3764c21a/attachment.txt.

[4] A. Buckley, J. Gibbons, and M. Reinhold. *State of the Lambda*. Sun Microsystem, Inc., October 2010. http://cr.openjdk.java.net/∼briangoetz/lambda/lambda-state-3.html.

[5] M. Bellia and M.E. Occhiuto. Higher Order Programming in Java: Introspection, Subsumption and Extraction. *Fundamenta Informaticae*, 67(1):29–44, 2005.

[6] M. Bellia and M.E. Occhiuto. Methods as Parameters: A Preprocessing Approach to Higher Order in Java. *Fundamenta Informaticae*, 85(1):35–50, 2008.

[7] M. Bellia and M.E. Occhiuto. *JavaΩ: Preprocessing Closures in Java*. University of Pisa, Dipartimento Informatica, 2009.

[8] G. Bracha, M. Odersky, D. Stoutamire, and P. Wadler. Making the Future Safe for the Past: Adding Genericity to the Java Programming Language. In *OOPSLA 1998*, pages 183–200. Sigplan, 1998.

[9] B. Bringert. HOJ - Higher-Order Java, 2005. http://www.cse.chalmers.se/alumni/bringert/hoj/.

[10] S. Colebourne. Closures in JDK7, 2009. http://www.jroller.com/scolebourne/entry/closures_in_jdk_7.

[11] Microsoft Corporation. Delegates in Visual J++, 2004. http://msdn.microsoft.com/en-us/library/aa260511(v=vs.60).aspx.

[12] S. Colebourne and S. Shulz. First_class methods: Java style closures, 2007. http://jroller.com/scolebourne/entry/first_class_methods_java_style.

[13] M. Felleisen D.P. Friedman. A Reduction Semantics for Imperative High-Order Languages. In *PARLE Parallel Architectures and Languages Europe*, volume 2007 of *LNCS*, pages 206–223. Springer, 1987.

[14] E. Gamma, R. Helm, R. Johnson, and J.M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 2005.

[15] N.M. Gafter and P. von der Ahè. Closures for Java (v0.6a), 2009. http://javac.info/closures-v06a.html.

[16] A. Igarashi and B. Pierce. On Inner Classes. *Information and Computation*, 77(1):56–89, 2002.

[17] A. Igarashi, B. Pierce, and P. Wadler. Featherweight Java: A Minimal Core Calculus for Java and GJ. *Sigplan Notices*, 34(10):132–146, 1999.

[18] A. Igarashi, B. Pierce, and P. Wadler. Featherweight Java: A Minimal Core Calculus for Java and GJ. *ACM TOPLAS*, 23:396–450, 2001.

[19] B. Lee, D. Lea, and J. Bloch. Concise Instance Creation Expressions: Closure without Complexity, 2006. http://crazybob.org/2006/10/java-closure-spectrum.html.

[20] E. Meijer. Lambada, Haskell as a better Java. *Electronic Notes TCS*, 41(1), 2001.

[21] Sun Microsystems. About Microsoft's Delegates, 2004. http://java.sun.com/docs/white/delegates.html.

[22] M. Odersky and P. Wadler. Pizza into Java: Translating Theory into Practice. In *Proc. 24th Symposium on Principles of Programming Languages*, pages 146–159, 1997.

[23] M. Reinhold. Project Lambda: Straw-Man Proposal, 2009. http://cr.openjdk.java.net/∼mr/lambda/straw-man/.

[24] M. Reinhold. There's not a Moment to Lose! - Closures for Java, 2009. http://blogs.sun.com/mr/entry/closures.

[25] A. Setzer. Java as a Functional Programming Language. In *TYPES 2002,LNCS 2646.*, pages 279–298, 2003.

[26] N. Sridranop and R. Stansifer. Higher-Order Functional Programming and Wildcards in Java. In *ACMSE 2007, ACM*, pages 42–46, 2007.