Soundness of object-oriented languages with coinductive big-step semantics*

Davide Ancona

DISI, Università di Genova, Italy
davide@disi.unige.it

Abstract. It is well known that big-step operational semantics are not suitable for proving soundness of type systems, because of their inability to distinguish stuck from non-terminating computations. We show how this problem can be solved by interpreting coinductively the rules for the standard big-step operational semantics of a Java-like language, thus making the claim of soundness more intuitive: whenever a program is well-typed, its coinductive operational semantics returns a value.
Indeed, coinduction allows non-terminating computations to return values; this is proved by showing that the set of proof trees defining the semantic judgment forms a complete metric space when equipped with a proper distance function.
In this way, we are able to prove soundness of a nominal type system w.r.t. the coinductive semantics. Since the coinductive semantics is sound w.r.t. the usual small-step operational semantics, the standard claim of soundness can be easily deduced.

1 Introduction

It is well known that standard big-step operational semantics are less amenable to prove soundness of type systems than small-step semantics; several important motivations for this statement can be found in the literature [12,13], but, basically, the main source of all problems is the inability of big-step operational semantics to distinguish stuck from non-terminating computations.

Besides addressing this problem, our work seeks to find simpler, and easy to be automated, techniques for proving soundness of abstract compilation of object-oriented languages [7,4,6,5], a novel approach which aims to reconcile type analysis and symbolic execution, where programs are compiled into a constraint logic program (CLP), and type analysis corresponds to solving a certain goal w.r.t. the coinductive semantics of CLP.

The idea of using coinduction to allow big-step semantics to capture non-terminating computations is not new (see the conclusion for a brief survey); Leroy

* This work has been partially supported by MIUR DISCO - Distribution, Interaction, Specification, Composition for Object Systems. I would like to thank Erik Ernst for his useful comments and suggestions; many thanks also to Sophia Drossopoulou, Atsushi Igarashi, Alan Mycroft and Elena Zucca for all their useful suggestions and questions.

and Grall [13] have investigated coinductive operational semantics in the context of functional programming, with the main aim of elaborating techniques for automatically proving the correctness of compilation. Among several approaches considered by the authors, the simplest one consists in interpreting coinductively the standard rules for the big-step operational semantics of lambda-calculus, and then expressing the soundness claim in a very direct way: if an expression e has type τ, then the coinductive semantics of e yields a value v of type τ. Unfortunately, such a claim fails to hold, as shown by the authors themselves, since there exist well-typed non-terminating expressions for which the coinductive semantics is not defined. This happens because only finite values are considered, whereas the values that should be returned by the coinductive semantics of such counter-example expressions correspond to necessarily infinite limits of sequences of finite (that is, inductively defined) values. More formally, if only finite values are considered, then it is not possible to define a complete metric space over the set of possibly infinite proof trees for the judgment of the coinductive semantics.

Interestingly enough, if the same approach is taken for a Java-like language, and, more importantly, infinite values are considered as well, then the claim of soundness holds when expressed in terms of a coinductive big-step semantics.

Figure 1 provides a road-map to the main defined judgments and proved claims in this paper. Symbols $\vdash$ and $\Vdash$ denote judgments defined inductively

$$\boxed{\Gamma \vdash e{:}\tau} \quad \overset{\text{Sect. 5}}{\Rightarrow} \quad \boxed{\Gamma \Vdash e{:}\tau} \quad \overset{\text{Sect. 6}}{\Rightarrow} \quad \boxed{\exists\, \mathtt{v}.\Pi \Vdash e \Rightarrow \mathtt{v}} \quad \overset{\text{Sect. 4}}{\Rightarrow} \quad \boxed{e \text{ diverges or } \exists\, v.e \xrightarrow{*} v}$$

Fig. 1. Relationship between the main judgments

and coinductively, respectively.

After having introduced the syntax and the standard small-step semantics (abbreviated with ISS) of the language (Section 2), and the mathematical background (Section 3) needed for the proofs, in Section 4 we define the coinductive big-step semantics (abbreviated with CBS) of the language, show, by means of examples, its behavior in case of non-termination, and formalize its relationship with the ISS (rightmost implication in Figure 1): if the CBS of e yields a value $\mathtt{v}$, then the ISS of e either returns a value v, or does not terminate; in other words, whenever the CBS of e yields a value, the ISS of e cannot get stuck, hence, the CBS is sound w.r.t. the ISS. Note the different nature of values (and hence the use of different meta-variables) in the CBS, where they can be infinite, and in the ISS, where they can only be finite.

In Section 5 a conventional inductive and nominal type system is defined (judgment $\Gamma \vdash e{:}\tau$), and a coinductive type system is derived from it (judgment $\Gamma \Vdash e{:}\tau$): such a coinductive system is closer to the CBS, indeed it can be regarded as an abstraction of the CBS. Furthermore we prove that all judgments that hold in the inductive type system can be derived in the coinductive one as well (leftmost implication in Figure 1).

The core and most difficult part of the formalization concerns the proof of the soundness of the coinductive type system w.r.t. the CBS (middle implication in Figure 1, proved in Section 6). The overview in Figure 1 clearly shows that, as expected, in the end we obtain the the standard soundness result expressed in terms of the ISS. At the end of the section we propose a scheme for proving soundness for a generic type system and language; the rightmost implication in Figure 1 is needed only if one wants to relate the CBS to the ISS, and derive from it a standard soundness claim expressed in terms of the ISS. Furthermore, such an implication needs to be proved just once per programming language, since it does not depend from the considered type system. We expect the definition of the coinductive type system to be given in terms of the inductive one, and the proof of the leftmost implication in Figure 1 to be standard, whereas the proof given in Section 6, with the corresponding definitions of metric spaces, should provide a template to be adapted for proving the middle implication in Figure 1.

Finally, in Section 7 we outline conclusions and related work.

This paper is an extension of a previous work by the same author [3], where the following contributions have been added: The definition of the coinductive type system and the corresponding proof of the leftmost implication in Figure 1 are new; the coinductive type system has been introduced to make the proof of soundness simpler and more modular; it also reveals how coinduction is related to the inductive type system.

While the justification for the introduction of infinite values in the CBS is only informally motivated in the previous work, here it has been made rigorous by means of the notion of complete metric space; in particular, the definition of the metric space of proof trees for the CBS judgment, and the proof of its properties, represent a non trivial task and an original contribution.

In Section 4 a new example has been added (case 2 (c)), showing that soundness does not hold if only infinite but regular values are considered.

All main proofs have been detailed. All omitted proofs and definitions can be found in the companion technical report[1].

2 Definition of the language

In this section we present our simple Java-like language, which will be used as reference language throughout the paper, together with its standard call-by-value small-step operational semantics.

Syntax: The syntax of the language is defined in Figure 2.

The language is a modest variation of Featherweight Java (FJ) [11], where the main differences concern the introduction of conditional expressions and boolean values, and the omission of type casts.

[1] Available at `ftp://ftp.disi.unige.it/person/AnconaD/ecoop12long.pdf`.

$$
\begin{aligned}
p &::= \overline{cd}^n\ e\\
cd &::= \textbf{class}\ c_1\ \textbf{extends}\ c_2\ \{\ \overline{fd}^n\ \overline{md}^k\ \}\quad (c_1 \neq \texttt{Object})\\
fd &::= \tau\ f;\\
md &::= \tau_0\ m(\overline{\tau\ x}^n)\ \{e\}\quad x_i \neq \texttt{this}\ \forall i = 1..n\\
\tau &::= c \mid \textbf{bool}\\
e &::= \textbf{new}\ c(\overline{e}^n) \mid x \mid e.f \mid e_0.m(\overline{e}^n) \mid \textbf{if}\ (e)\ e_1\ \textbf{else}\ e_2\\
&\quad \mid \textbf{false} \mid \textbf{true}
\end{aligned}
$$

Assumptions: $n, k \geq 0$, inheritance is acyclic, names of declared classes in a program, methods and fields in a class, and parameters in a method are distinct.

Fig. 2. Syntax of the language

Standard syntactic restrictions are implicitly imposed in the figure. Bars denote sequences of n items, where n is the superscript of the bar and the first index is 1. Sometimes this notation is abused, as in $\overline{f}^h = \overline{e'}^h$; which is a shorthand for $f_1 = e'_1; \ldots f_h = e'_h;$.

A program consists of a sequence of class declarations and a main expression. Types can only be class names and the primitive type **bool**; we assume that the language supports boxing conversions, hence *bool* is a subtype[2] of the predefined class `Object`, which is the top type.

A class declaration contains field and method declarations; in contrast with FJ, constructors are not declared, but every class is equipped with an implicit constructor with parameters corresponding to all fields, in the same order as they are inherited and declared. For instance, the classes defined below

```
class P extends Object{bool b; P p;}
class C extends P{C c;}
```

have the following implicit constructors:

```
P(bool b,P p){super(); this.b=b; this.p=p;}
C(bool b,P p,C c){super(b,p); this.c=c;}
```

Method declarations are standard; in the body, the target object can be accessed via the implicit parameter `this`, therefore all explicitly declared formal parameters must be different from `this`. Expressions include instance creation, variables, field selection, method invocation, conditional expressions, and boolean literals.

Small-step operational semantics The definition of the conventional small-step operational semantics of the language can be found in Figure 3. We follow the approach of FJ, even though for simplicity we have preferred to restrict the semantics to the deterministic call-by-value evaluation strategy.

[2] This assumption ensures the existence of the join between types, without introducing union types, to make the typing rule for conditional expressions simpler in the type system defined in Section 5.

Values are either the literals **false** or **true**, or object expressions in normal form having shape **new** $c(\overline{v}^n)$. As happens for FJ, the semantics of object creation is more liberal than the expected one; indeed, **new** $c(\overline{v}^n)$ is always a correct expression which reduces to itself in zero steps, even when class c is not declared, or the number of arguments does not match the number of fields of c. As we will see, the big-step semantics follows a less liberal semantics, more in accordance with the standard semantics of mainstream object-oriented languages.

As usual, the reduction relation $\rightarrow$ should be indexed over the collection of all class declarations contained in the program (conventionally called class table), however for brevity we leave implicit such an index in all judgments defined in the paper. The reflexive and transitive closure of $\rightarrow$ is denoted by $\stackrel{*}{\rightarrow}$.

The definition of the standard auxiliary functions *fields* and *meth* is straightforward (see the companion technical report). For compactness, such functions provide semantic and type information at once, since they are instrumental for the definition of both the semantics and the type system of the language. Function *fields* returns the list of all fields which are either inherited or declared in the class, in the standard order and with the corresponding declared types. In the case of the predefined class `Object` the returned list is empty (ϵ); field hiding is not supported, hence *fields* is not defined if a class declares a field with the same name of an inherited one. Function *meth* performs standard method look-up: if $meth(c, m) = \overline{\tau}^n\ \overline{x}^n.e{:}\tau$, then look-up of method m starting from class c returns the corresponding declaration where $\overline{\tau}^n\ \overline{x}^n$ are the formal parameters with their declared types, and e and τ are the body and the declared returned type, respectively. If $meth(c, m)$ is undefined, then it means that look-up of m from c fails.

In rule (fld), if f_i is a field of the class, then the expression reduces to the corresponding value passed to the implicit constructor. If the selected field is not in such a list, then the evaluation of the expression gets stuck.

In rule (inv), if method look-up succeeds starting from the class of the target object, then the corresponding body is executed, where the implicit parameter `this` and the formal parameters are substituted with the target object and the argument values, respectively. The notation $e_i[\overline{x}^n \mapsto \overline{v}^n]$ denotes parallel substitution of the distinct variables $\overline{x}^n$ with values $\overline{v}^n$ in the expression e.

Rules for conditional expressions (ift) and (iff), and for context closure (ctx) are straightforward. Contexts are the standard ones corresponding to left-to-right, call-by-value strategy.

3 Background

In the following, with the term *tree* over a set S we will mean a finitely branching tree with nodes in S that is allowed to contain infinite paths. If t is a tree, we denote with $root(t)$ the root of t.

More rigorously, a tree with infinite paths can be defined in terms of partial functions over finite paths of natural numbers (denoted by $\mathbb{N}^*$) [9,7].

$$v ::= \mathbf{new}\ c(\overline{v}^n) \mid \mathbf{false} \mid \mathbf{true}$$
$$\mathcal{C}[\,] ::= \square \mid \mathbf{new}\ c(\overline{v}^n, \square, \overline{e}^k) \mid \square.f \mid \square.m(\overline{e}^n) \mid v.m(\overline{v}^n, \square, \overline{e}^k) \mid \mathbf{if}\ (\square)\ e_1\ \mathbf{else}\ e_2$$

$$\text{(fld)}\dfrac{\mathit{fields}(c) = \overline{\tau}^n\ \overline{f}^n, \quad 1 \leq i \leq n}{\mathbf{new}\ c(\overline{v}^n).f_i \to v_i}$$

$$\text{(inv)}\dfrac{\mathit{meth}(c, m) = \overline{\tau}^n\ \overline{x}^n.e{:}\tau}{\mathbf{new}\ c(\overline{v}^k).m(\overline{v'}^n) \to e[\mathtt{this} \mapsto \mathbf{new}\ c(\overline{v}^k), \overline{x}^n \mapsto \overline{v'}^n]}$$

$$\text{(ift)}\dfrac{}{\mathbf{if}\ (\mathbf{true})\ e_1\ \mathbf{else}\ e_2 \to e_1} \qquad \text{(iff)}\dfrac{}{\mathbf{if}\ (\mathbf{false})\ e_1\ \mathbf{else}\ e_2 \to e_2} \qquad \text{(ctx)}\dfrac{e \to e'}{\mathcal{C}[e] \to \mathcal{C}[e']}$$

Fig. 3. Call-by-value inductive small-step operational semantics

Definition 1. *A tree over a set S is a partial function $t : \mathbb{N}^* \to S$ satisfying the following properties:*

1. *its domain is not empty: $dom(t) \neq \emptyset$;*
2. *its domain is prefix-closed: $p \cdot n \in dom(t)$ implies $p \in dom(t)$, for all $p \in \mathbb{N}^*$, and $n \in \mathbb{N}$;*
3. *if $p \cdot n \in dom(t)$ and $k \leq n$, then $p \cdot k \in dom(t)$ for all $p \in \mathbb{N}^*$ and $n, k \in \mathbb{N}$;*
4. *there exists $n \in \mathbb{N}$ s.t. $p \cdot n \notin dom(t)$, for all $p \in \mathbb{N}^*$.*

Every path $p \in dom(t)$ identifies a unique subtree t' of t whose root is $t(p)$: $dom(t') = \{p' \in \mathbb{N}^* \mid p \cdot p' \in dom(t)\}$, and $t'(p') = t(p \cdot p')$ for all $p' \in dom(t')$.

Definition 2. *A* regular *(a.k.a. rational) tree is a tree whose set of subtrees is finite.*

Trivially, every finite tree (that is, tree with only finite paths) is regular, but there exist also infinite trees that are regular.

Definition 3. *A metric space (S, d) is a set S equipped with a function $d{:}S \times S \to \mathbb{R}$, called* metric *or* distance*, satisfying the following properties, for all x, y, and z in S:*

- *(identity) $d(x, y) = 0$ iff $x = y$;*
- *(symmetry) $d(x, y) = d(y, x)$;*
- *(triangle inequality) $d(x, z) \leq d(x, y) + d(y, z)$.*

Definition 4. *Let (S, d) be a metric space.*

- *A sequence $(x_i)_{i \in \mathbb{N}}$ has limit l iff for all $\epsilon > 0$ there exists $k \in \mathbb{N}$ s.t. $d(x_n, l) < \epsilon$, for all $n > k$.*
- *A Cauchy sequence $(x_i)_{i \in \mathbb{N}}$ is a sequence s.t. for all $\epsilon > 0$ there exists $k \in \mathbb{N}$ s.t. $d(x_n, x_m) < \epsilon$ for all $n, m > k$.*
- *A metric space is* complete *iff all Cauchy sequences have a limit.*

Proposition 1. *Let T_S be the set of all trees over S. Then, T_S is a complete metric space [8,2] with the following metric:*

$$d_T(t_1, t_2) = 2^{-c}$$

where $c = shtp(t_1, t_2) = \min\{n \in \mathbb{N} \mid p \in \mathbb{N}^n, t_1(p) \neq_\perp t_2(p)\}$, $min\,\emptyset = \infty$, $2^{-\infty} = 0$, $t_1(p) =_\perp t_2(p)$ iff either $p \notin dom(t_1)$ and $p \notin dom(t_2)$, or $p \in dom(t_1) \cap dom(t_2)$ and $t_1(p) = t_2(p)$. That is, c is the length of a shortest path that distinguishes t_1 from t_2, if $t_1 \neq t_2$, or $c = \infty$ if $t_1 = t_2$.

By definition, for all pairs of trees t_1 and t_2, $d_T(t_1, t_2) \in \{0\} \cup \{2^{-c} \mid c \in \mathbb{N}\}$, that is, $0 \leq d_T(t_1, t_2) \leq 1$. It can be proved that the set of finitely branching trees with infinite paths, with the metric defined above, is the (unique up to isometries) completion of the set of finitely branching trees with finite paths with the same metric.

Definition 5. *Let us consider a judgment where all possible instantiations range over the set $\mathcal{J}$.*

A proof tree *∇ for an instantiation $j \in \mathcal{J}$ of the judgment is a tree ∇ over $\mathcal{J}$ s.t. root$(\nabla) = j$.*

A valid proof tree *for an instantiation $j \in \mathcal{J}$ of the judgment is a proof tree ∇ s.t. for any node j in ∇, if $j_1, \ldots, j_k$ are the children of j, then $\frac{j_1, \ldots, j_k}{j}$ is a correct instantiation of one of the meta-rules defining this kind of judgment.*

We simply write that ∇ is a (valid) proof tree for the judgment, when we are not interested in specifying the particular instantiation $j \in \mathcal{J}$ which is the root of the tree.

We write $ok(\text{R})(\nabla)$ to indicate that the root of ∇ together with its children are a correct instantiation of the meta-rule labeled by R. Hence, a valid proof tree ∇ is a proof tree s.t. for all subtrees ∇' of ∇ (including ∇), there exists a meta-rule R s.t. $ok(\text{R})(\nabla')$.

Definition 6. *A* complete lattice *is a partially ordered set $(L, \leq)$ s.t. any subset S of L has a supremum (a.k.a. least upper bound) denoted with* $\sup S$.

Since, by definition of inf and sup, $\inf S = \sup\{x \in L | \forall\, y \in S.x \leq y\}$, $\sup \emptyset$ is the least element $\perp$ of L, and $\inf \emptyset$ is the greatest element $\top$ of L, then every subset of a complete lattice has an infimum (greatest lower bound) and every complete lattice is bounded.

Definition 7. *Let $(L, \leq)$ be a complete lattice. A (total) function $f : L \to L$ is* continuous *iff it preserves the supremum and infimum of every subset of L: for all $S \subseteq L$, $f(\sup S) = \sup\{f(x) | x \in S\}$ and $f(\inf S) = \inf\{f(x) | x \in S\}$.*

It is easy to prove that a continuous function is monotone.

Definition 8. *Let $(L, \leq)$ be a partially ordered set, f a (total) function from L to L, and x an element of L.*

- *x is a pre-fixed point of f (a.k.a. f-closed) iff $f(x) \leq x$;*
- *x is a post-fixed point of f (a.k.a. f-dense or f-justified or f-consistent) iff $x \leq f(x)$.*

Trivially, x is a fixed-point of f iff x is both a pre-fixed and a post-fixed point of f.

Theorem 1 (Tarski-Knaster). *Let $(L, \leq)$ be a complete lattice, and $f : L \to L$ a monotone function. Then*

1. *$f(\inf\{x \in L | x \text{ pre-fixed point of } f\}) = \inf\{x \in L | x \text{ pre-fixed point of } f\}$;*
2. *$f(\sup\{x \in L | x \text{ post-fixed point of } f\}) = \sup\{x \in L | x \text{ post-fixed point of } f\}$.*

From Theorem 1 one can trivially deduce that a monotone function defined on a complete lattice has always a least fixed-point (which is also the least pre-fixed point), and a greatest fixed point (which is also the greatest post-fixed point).

Given a judgment defined by a set of meta-rules, with instantiations ranging over $\mathcal{J}$, it is possible to define the one step inference function $\mathcal{F}$ over the powerset of $\mathcal{J}$ as follows: for any subset J of $\mathcal{J}$, $\mathcal{F}(J)$ is the subset J' of $\mathcal{J}$ s.t. for any $j \in J'$, there exists a correct instantiation $\frac{j_1,\ldots,j_k}{j}$ of a meta-rule with $\{j_1, \ldots, j_k\} \subseteq J$.

Such a function is always trivially monotone, and it can be proved [7,5,13,17] that its least fixed point is the set of $j \in \mathcal{J}$ s.t. there exists a finite valid proof tree for j, whereas its greatest fixed point is the set of $j \in \mathcal{J}$ s.t. there exists a (possibly infinite) valid proof tree for j.

We denote with $f^n(x)$ n iterated applications of f to x (with $n \in \mathbb{N}$, $f^0(x) = x$).

Theorem 2 (Kleene). *Let $(L, \leq)$ be a complete lattice, and $f : L \to L$ a continuous function. Then*

1. *$\sup\{f^n(\bot) | n \in \mathbb{N}\}$ is the least fixed point of f;*
2. *$\inf\{f^n(\top) | n \in \mathbb{N}\}$ is the greatest fixed point of f.*

Since f is monotone, we have that $f^0(\bot) \leq f^1(\bot) \leq \ldots f^n(\bot) \leq f^{n+1}(\bot) \leq \ldots$ is an ascending chain, and dually, $f^0(\top) \geq f^1(\top) \geq \ldots f^n(\top) \geq f^{n+1}(\top) \geq \ldots$ is a descending chain. Note that the two claims of Theorem 2 hold also under the weaker assumption that f is a monotone function preserving suprema of ascending chains (claim 1), or infima of descending chains (claim 2).

4 A coinductive semantics

In this section we define a call-by-value coinductive big-step operational semantics for our language.

Such a semantics is obtained by simply interpreting coinductively the definition of values and the rules of the standard inductive big-step operational semantics (with no rules for error handling).

Definition of the semantics The CBS judgment uses value environments (see below), just for uniformity with the type judgment defined in Section 5. Value environments are not strictly necessary, since the rule for method invocation can be equivalently defined with parallel substitution as in ISS. Values are separated from expressions since they are infinite, while expressions are always finite. Such a separation is further stressed by the fact that values belong to a different syntactic category, that is, even finite values are different from expressions.

$$\mathtt{v}, \mathtt{u} ::= \mathit{obj}(c, [\overline{f}^n \mapsto \overline{\mathtt{v}}^n]) \mid \mathit{false} \mid \mathit{true} \quad (\mathit{coinductive\ def.})$$

We recall that **false** and **true** are expressions of our language, and values (denoted by the meta-variable v) in the ISS, whereas *false* and *true* are not expressions, but just the corresponding values (denoted by the meta-variable $\mathtt{v}$) in the CBS. Similarly, **new** $c(\mathbf{true})$ is both an expression and a value in ISS, whereas $\mathit{obj}(c, [f \mapsto \mathit{true}])$ is the corresponding value in CBS (assuming that the only field of c is f), and is not an expression.

As an example of an infinite value, let us consider the object value $\mathtt{v}$ defined by the equation

$$\mathtt{v} = \mathit{obj}(\mathit{List}, [\mathit{hd} \mapsto \mathit{obj}(\mathit{Elem}, [\,]), \mathit{tl} \mapsto \mathtt{v}])$$

which represents an infinite list; in our language, such a value can only be returned by an infinite computation. Of course in a lazy or imperative language, this value could be returned also by a terminating computation; however, the important point here is that type correct expressions which do not terminate must always return a value in the CBS: as explained in case 2 in the second part of this section, without infinite values the claim of soundness proved in Section 5 would not hold.

The CBS is defined in Figure 4. Thicker lines manifest that rules are interpreted coinductively. A value environment Π is a finite sequence $\overline{x_i}^n \mapsto \overline{\mathtt{v}}^n$, where all variables $\overline{x_i}^n$ are distinct, denoting a finite partial function mapping variables to values ($\emptyset$ denotes the empty environment, $\mathit{dom}(\Pi)$ the domain of Π). Environments model stack frames of method invocations.

Rules (VAR), (FAL), and (TRU) are straightforward. Evaluation of instance creation (NEW) succeeds only if $\mathit{fields}(c)$ is defined (that is, if c and its ancestors are declared in the program and no field is hidden), and returns a list of fields whose length must coincide with the number of arguments; then all arguments are evaluated and the obtained values are associated with the corresponding fields in the object value. For field selection (FLD) the target expression is evaluated; then evaluation succeeds only if an object value is returned, and the selected field is present in the object value; in this case the corresponding associated value is returned. For method invocation (INV) all expressions denoting the target object and the arguments are evaluated. If the value corresponding to the target is an object of class c, method look-up starting from c succeeds and returns a method declaration with a number of formal parameters coinciding with the number of passed arguments, then the method body is evaluated in the environment where `this` and the formal parameters are associated with their corresponding values. If such an evaluation succeeds, then the returned value is the value of the method invocation. Finally, rules (IFT) and (IFF) deal with the straightforward evaluation of conditional expressions.

$$\text{(VAR)}\ \frac{\Pi(x) = \mathtt{v}}{\Pi \Vdash x \Rightarrow \mathtt{v}} \qquad \text{(FAL)}\ \frac{}{\Pi \Vdash \textbf{false} \Rightarrow \mathit{false}} \qquad \text{(TRU)}\ \frac{}{\Pi \Vdash \textbf{true} \Rightarrow \mathit{true}}$$

$$\text{(NEW)}\ \frac{\forall\, i = 1..n. \Pi \Vdash e_i \Rightarrow \mathtt{v}_i \quad \mathit{fields}(c) = \overline{\tau}^n\ \overline{f}^n}{\Pi \Vdash \textbf{new}\ c(\overline{e}^n) \Rightarrow \mathit{obj}(c, [\overline{f}^n \mapsto \overline{\mathtt{v}}^n])} \qquad \text{(IFT)}\ \frac{\Pi \Vdash e \Rightarrow \mathit{true} \qquad \Pi \Vdash e_1 \Rightarrow \mathtt{v}}{\Pi \Vdash \textbf{if}\ (e)\ e_1\ \textbf{else}\ e_2 \Rightarrow \mathtt{v}}$$

$$\text{(IFF)}\ \frac{\Pi \Vdash e \Rightarrow \mathit{false} \qquad \Pi \Vdash e_2 \Rightarrow \mathtt{v}}{\Pi \Vdash \textbf{if}\ (e)\ e_1\ \textbf{else}\ e_2 \Rightarrow \mathtt{v}} \qquad \text{(FLD)}\ \frac{\Pi \Vdash e \Rightarrow \mathit{obj}(c, [\overline{f}^n \mapsto \overline{\mathtt{v}}^n]) \quad 1 \leq i \leq n}{\Pi \Vdash e.f_i \Rightarrow \mathtt{v}_i}$$

$$\text{(INV)}\ \frac{\begin{array}{l}\forall\, i = 0..n. \Pi \Vdash e_i \Rightarrow \mathtt{v}_i \quad \mathtt{this} \mapsto \mathtt{v}_0, \overline{x}^n \mapsto \overline{\mathtt{v}}^n \Vdash e \Rightarrow \mathtt{v} \\ \mathtt{v}_0 = \mathit{obj}(c, [\ldots]) \quad \mathit{meth}(c, m) = \overline{\tau}^n\ \overline{x}^n.e{:}\tau\end{array}}{\Pi \Vdash e_0.m(\overline{e}^n) \Rightarrow \mathtt{v}}$$

Fig. 4. Call-by-value coinductive big-step operational semantics

Note that in the CBS, object creation is less liberal than in the ISS: as an example, **new** c`()` is a value in the ISS, whereas the same expression may not evaluate to a value in the CBS; this happens if either c is not declared in the program, or if c contains at least one field.

Coinductive semantics of non-terminating expressions We have already observed that if the definition of values and the evaluation rules are interpreted inductively, then we obtain a standard inductive big-step operational semantics. Obviously, if an expression evaluates to a value in the inductive semantics, then the same value is obtained in the coinductive one; however, this case concerns terminating expressions, whereas what we do really care about here is the behavior of the CBS for non-terminating expressions. We show that three different cases may occur. All expressions e considered in the examples below are well-typed and do not terminate in the ISS, that is, there exists no normal form e' s.t. $e \xrightarrow{*} e'$.

Case 1: There exist many values $\mathtt{v}$ s.t. $\emptyset \Vdash e \Rightarrow \mathtt{v}$

Let us consider the expression $e =$ **new** `C().m()`, where `C` is the only class declared in the program:

```
class C extends Object{bool m(){this.m()}}
```

Then $\emptyset \Vdash e \Rightarrow \mathtt{v}$ for all values $\mathtt{v}$, as shown in the valid proof tree of Figure 5. Ellipsis means that such a tree is infinite (hence, it cannot be a valid proof for an inductive system), although regular, that is, it can be folded into a finite graph, because of the repeated finite pattern originated from the judgment

$\Pi\Vdash\mathtt{this.m()} \Rightarrow \mathbb{v}$. Such non-determinism is naturally reflected in the conven-

$$\dfrac{\dfrac{}{\emptyset\Vdash\mathbf{new}\ \mathtt{C()} \Rightarrow \mathbb{u}} \qquad \dfrac{\dfrac{}{\Pi\Vdash\mathtt{this} \Rightarrow \mathbb{u}} \quad \dfrac{\vdots}{\Pi\Vdash\mathtt{this.m()} \Rightarrow \mathbb{v}}}{\Pi\Vdash\mathtt{this.m()} \Rightarrow \mathbb{v}}}{\emptyset\Vdash\mathbf{new}\ \mathtt{C().m()} \Rightarrow \mathbb{v}}$$

Fig. 5. Proof tree for $\emptyset\Vdash e \Rightarrow \mathbb{v}$, where $\mathbb{u} = \mathit{obj}(\mathtt{C}, [\,])$, $\Pi = \mathtt{this} \mapsto \mathbb{u}$

tional nominal type system (see Section 5) where the return type **bool** can in fact be correctly replaced by any other type defined in the program.

There are also cases where finitely many values are returned. For instance,

$$\emptyset\Vdash\mathbf{if}\,\mathtt{(}\mathbf{new}\ \mathtt{C().m())}\ \mathbf{true\ else\ false} \Rightarrow \mathit{true}$$
$$\emptyset\Vdash\mathbf{if}\,\mathtt{(}\mathbf{new}\ \mathtt{C().m())}\ \mathbf{true\ else\ false} \Rightarrow \mathit{false}$$

and no other values can be returned.

Case 2 (a), (b) and (c): There exists a unique value $\mathbb{v}$ s.t. $\emptyset\Vdash e \Rightarrow \mathbb{v}$

We consider three possible cases (a), (b), and (c), where the returned value is finite (a), or infinite but regular (b), or infinite and non regular (c). For case (a), if `C` is the class of case 1, then the expression **if(new** `C().m())` **true else true** trivially evaluates to the unique value *true* (although with two different valid proof trees). For case (b), let us consider a program with the following declarations (where `M`, `L`, and `n` are abbreviations for `Main`, `List`, and `next`, respectively):

```
class M extends Object{L m(){new L(this.m())}}
class L extends Object{L n;}
```

The main expression **new** `M().m()` evaluates to a unique value which is an infinite but regular object of class `L`; Figure 6 shows the unique valid proof tree for $\emptyset\Vdash\mathbf{new}\ \mathtt{M().m()} \Rightarrow \mathit{obj}(\mathtt{L}, [\mathtt{n} \mapsto \mathbb{v}])$; such a tree is infinite, but regular. The proof tree is valid if and only if the following proposition holds:

$$\Pi\Vdash\mathbf{new}\ \mathtt{L(this.m())} \Rightarrow \mathbb{v} \text{ iff } \Pi\Vdash\mathbf{new}\ \mathtt{L(this.m())} \Rightarrow \mathit{obj}(\mathtt{L}, [\mathtt{n} \mapsto \mathbb{v}])$$

with $\Pi = \mathtt{this} \mapsto \mathit{obj}(\mathtt{M}, [\,])$. Such a proposition cannot be satisfied by finite values, but holds for the unique infinite regular value $\mathbb{v}$ s.t. $\mathbb{v} = \mathit{obj}(\mathtt{L}, [\mathtt{n} \mapsto \mathbb{v}])$.

In the conventional nominal type system the return type τ of method m in `M` must verify $\mathtt{L} \leq \tau$, since the body of the method returns a new instance of class `L`, but also $\tau \leq \mathtt{L}$, since the formal parameter of the implicit constructor of `L` has the same type as field `n`; therefore, similarly to what happens in the CBS, there exists only one possible return type: `L`. This example shows that if rules are interpreted coinductively, but values can only be finite, then the soundness claim proved in Section 5, (that is, any well-typed expression evaluates to a value) does not hold.

Finally, for case (c), let us consider the following class declarations:

$$
\dfrac{\dfrac{}{\emptyset \Vdash \mathbf{new}\ \mathtt{M()} \Rightarrow \mathit{obj}(\mathtt{M}, [\,])} \qquad \dfrac{\dfrac{\dfrac{}{\Pi \Vdash \mathtt{this} \Rightarrow \mathit{obj}(\mathtt{M}, [\,])} \qquad \dfrac{\vdots}{\Pi \Vdash \mathbf{new}\ \mathtt{L(this.m())} \Rightarrow \mathbb{v}}}{\Pi \Vdash \mathtt{this.m()} \Rightarrow \mathbb{v}}}{\Pi \Vdash \mathbf{new}\ \mathtt{L(this.m())} \Rightarrow \mathit{obj}(\mathtt{L}, [\mathtt{n} \mapsto \mathbb{v}])}}{\emptyset \Vdash \mathbf{new}\ \mathtt{M().m()} \Rightarrow \mathit{obj}(\mathtt{L}, [\mathtt{n} \mapsto \mathbb{v}])}
$$

Fig. 6. Proof of $\emptyset \Vdash \mathbf{new}\ \mathtt{M().m()} \Rightarrow \mathit{obj}(\mathtt{L}, [\mathtt{n} \mapsto \mathbb{v}])$, with $\mathbb{v} = \mathit{obj}(\mathtt{L}, [\mathtt{n} \mapsto \mathbb{v}])$, $\Pi = \mathtt{this} \mapsto \mathit{obj}(\mathtt{M}, [\,])$

```
class Nat extends Object{ }
class Z extends Nat{ }
class NZ extends Nat{Nat p;}
class M extends Object{L m(Nat e){new L(e,this.m(new NZ(e)))}}
class L extends Object{Nat e; L n;}
```

Then, the expression **new** `M().m(`**new** `Z())` is well-typed and evaluates to the unique infinite and non regular value $\mathbb{v}_0$ where

$$
\begin{array}{l}
\mathbb{v}_i = \mathit{obj}(\mathtt{L}, [\mathtt{e} \mapsto \mathbb{u}_i, \mathtt{n} \mapsto \mathbb{v}_{i+1}]) \text{ for all } i \in \mathbb{N} \\
\mathbb{u}_0 = \mathit{obj}(\mathtt{Z}, [\,]) \qquad \mathbb{u}_i = \mathit{obj}(\mathtt{NZ}, [\mathtt{p} \mapsto \mathbb{u}_{i-1}]) \text{ for all } i \in \mathbb{N} \setminus \{0\}
\end{array}
$$

Figure 7 shows the non regular valid proof tree for the judgment, defined in terms of a countably infinite set of equations whose solutions are valid proof trees ${}_{\Rightarrow}\nabla_i$, for all $i \in \mathbb{N}$. This example shows that if rules are interpreted coinductively, but values can only be regular, then the soundness claim proved in Section 5, (that is, any well-typed expression evaluates to a value) does not hold.

$$
{}_{\Rightarrow}\nabla_i = \dfrac{\dfrac{}{\Pi_i \Vdash \mathtt{e} \Rightarrow \mathbb{u}_i} \qquad \dfrac{\dfrac{}{\Pi_i \Vdash \mathtt{this} \Rightarrow \mathit{obj}(\mathtt{M}, [\,])} \qquad \dfrac{\dfrac{}{\Pi_i \Vdash \mathtt{e} \Rightarrow \mathbb{u}_i}}{\Pi_i \Vdash \mathbf{new}\ \mathtt{NZ(e)} \Rightarrow \mathbb{u}_{i+1}} \qquad {}_{\Rightarrow}\nabla_{i+1}}{\Pi_i \Vdash \mathtt{this.m(}\mathbf{new}\ \mathtt{NZ(e))} \Rightarrow \mathbb{v}_{i+1}}}{\Pi_i \Vdash \mathbf{new}\ \mathtt{L(e,this.m(}\mathbf{new}\ \mathtt{NZ(e)))} \Rightarrow \mathbb{v}_i}
$$

$$
\dfrac{\dfrac{}{\emptyset \Vdash \mathbf{new}\ \mathtt{M()} \Rightarrow \mathit{obj}(\mathtt{M}, [\,])} \qquad \dfrac{}{\emptyset \Vdash \mathbf{new}\ \mathtt{Z()} \Rightarrow \mathbb{u}_0} \qquad {}_{\Rightarrow}\nabla_0}{\emptyset \Vdash \mathbf{new}\ \mathtt{M().m(}\mathbf{new}\ \mathtt{Z())} \Rightarrow \mathbb{v}_0}
$$

Fig. 7. Proof of $\emptyset \Vdash \mathbf{new}\ \mathtt{M().m(}\mathbf{new}\ \mathtt{Z())} \Rightarrow \mathbb{v}_0$, with $\Pi_i = \mathtt{this} \mapsto \mathit{obj}(\mathtt{M}, [\,]), \mathtt{e} \mapsto \mathbb{u}_i$ for all $i \in \mathbb{N}$

Case 3: There exist no values $\mathtt{v}$ s.t. $\emptyset \Vdash e \Rightarrow \mathtt{v}$

If `C` is as in case 1 (that is, **new** `C().m()` does not terminate), then the expression **if**(**new** `C().m()`) **true**`.m()` **else** **true**`.m()` does not evaluate to any value; this is a direct consequence of the fact that no rules are applicable for the expression **true**`.m()` since **true** does not evaluate to an object value. The main difference with the previous two cases is that here the expression to be evaluated cannot be typed in any type system insensitive to non-termination. Indeed, in the conventional nominal type system defined in Section 5 all examples except for this are well-typed. This example shows the main difference between the ISS and the CBS: in the former, there exist ill-typed expressions whose evaluation does not terminate (that is, does not get stuck), whereas in the latter all ill-typed expressions do not evaluate to a value.

Soundness of CBS w.r.t. ISS We prove now that the CBS is sound w.r.t.the ISS. More precisely, if $\emptyset \Vdash e \Rightarrow \mathtt{v}$, then in the ISS either e diverges (that is, e does not reduce to a normal form), or e reduces in zero or more steps to a value v s.t. $\emptyset \Vdash v \Rightarrow \mathtt{v}$. In other words, we are guaranteed that the evaluation of an expression will never get stuck in the ISS whenever it returns a value in the CBS. Under this point of view the CBS plays a role similar to that of a type system; indeed, to prove this property we use the standard proof technique based on the progress and subject reduction properties. Such a property tells us an important fact: type soundness of a type system can be equivalently proved in terms of the CBS, instead of the ISS. If soundness holds in terms of the CBS, then it holds in terms of the ISS as well, by virtue of the soundness property of the CBS w.r.t. the ISS we are going to prove.

The progress and subject reduction properties can be proved routinely (see the companion technical report), the former by induction on e, the latter by induction on the rules defining ISS. Proof by coinduction is only needed for the substitution lemma.

Theorem 3 (Progress). *If $\emptyset \Vdash e \Rightarrow \mathtt{v}$, then either e is a value, or there exists e' s.t. $e \rightarrow e'$.*

Subject reduction relies on the following restricted form of substitution lemma which suffices for proving Theorem 4.

Lemma 1 (Substitution). *If $\overline{x}^n \mapsto \overline{\mathtt{v}}^n \Vdash e \Rightarrow \mathtt{v}$, and for all $i = 1..n$ $\emptyset \Vdash v_i \Rightarrow \mathtt{v}_i$, then $\emptyset \Vdash e[\overline{x}^n \mapsto \overline{v}^n] \Rightarrow \mathtt{v}$.*

Theorem 4 (Subject reduction). *If $\emptyset \Vdash e \Rightarrow \mathtt{v}$, and $e \rightarrow e'$, then $\emptyset \Vdash e' \Rightarrow \mathtt{v}$.*

Corollary 1. *If $\emptyset \Vdash e \Rightarrow \mathtt{v}$, $e \xrightarrow{*} e'$, and e' is a normal form, then e' is a value, and $\emptyset \Vdash e' \Rightarrow \mathtt{v}$.*

Proof. By induction on the number n of steps needed to reduce e to e'. If $n = 0$, then $e = e'$, and trivially $\emptyset \Vdash e' \Rightarrow \mathtt{v}$; furthermore, since e' is a normal form, by progress (Theorem 3) e' is a value. If $n > 0$, then there exists e'' s.t. $e \rightarrow e''$, and e'' reduces to e' in $n - 1$ steps. By subject reduction (Theorem 4) $\emptyset \Vdash e'' \Rightarrow \mathtt{v}$, then we conclude by inductive hypothesis.

5 Type systems

To make the proof of soundness simpler and more modular, we first define a standard inductive nominal type system for our reference language, and then we derive from it a coinductive nominal type system, and prove that if an expression is well-typed in the inductive type system, than it is assigned the same type in the coinductive one. In other words, the inductive type system is sound w.r.t. the coinductive one; we conjecture that in fact the two systems are equivalent (hence, the coinductive system is sound w.r.t. the inductive one as well), but here we prove only the only implication we are interested in. In this way, soundness of the inductive type system in terms of the CBS can be directly derived from soundness of the coinductive type system in terms of the CBS (prove in Section 6).

Auxiliary definitions Besides functions *fields* and *meth*, already used for defining both the ISS and the CBS, the typing rules are based on the following auxiliary functions/operators, whose definition is straightforward (see the companion technical report). The standard subtyping relation $\leq$ between nominal types; the *override* predicate s.t. $\mathit{override}(c, m, \overline{\tau}^n, \tau)$ holds iff $\mathit{meth}(c', m)$ is undefined or $\mathit{meth}(c', m) = \overline{\tau'}^n\ \overline{x}^n.e{:}\tau'$, $\overline{\tau'}^n \leq \overline{\tau}^n$, and $\tau \leq \tau'$, with c' direct superclass of c; the join operator $\vee$ which computes the least upper bound $\vee(\tau_1, \tau_2)$ of two types τ_1 and τ_2 (this is always defined since inheritance is single, and *bool* is a subtype of the top type `Object`).

Typing rules The typing rules, which can be found in Figure 8, are quite standard. A type environment Γ is a finite sequence $\overline{x_i}^n{:}\overline{\tau}^n$, where all variables $\overline{x_i}^n$ are distinct, denoting a finite function mapping variables to types ($\emptyset$ denotes the empty type environment, $\mathit{dom}(\Gamma)$ the domain of Γ). Rules (*pro*), (*cla*), and (*met*) define well-typed programs, classes, and methods, respectively. The other rules define well-typed expressions w.r.t. a given type environment. Let us recall that, similarly to what happens for the operational semantics, all typing judgments are implicitly indexed over a class table containing all needed information on the classes declared in the program.

Membership relation To prove soundness of the type system w.r.t. the CBS, we first define a relation $\mathtt{v} \in \tau$ between the CBS values and nominal types: intuitively, such a relation defines the intended semantics of types as set of values [6]. Such a relation is coinductively defined by the following rules:

$$\text{(TOP)}\ \frac{}{\mathtt{v} \in \mathtt{Object}} \qquad \text{(BOOL)}\ \frac{\mathtt{v} = \mathit{false} \text{ or } \mathtt{v} = \mathit{true}}{\mathtt{v} \in \mathit{bool}}$$

$$\text{(OBJ)}\ \frac{\forall\, i = 1..n. \mathtt{v}_i \in \tau_i \quad c \leq c' \quad \mathit{fields}(c) = \overline{\tau}^n\ \overline{f}^n}{\mathit{obj}(c, [\overline{f}^n \mapsto \overline{\mathtt{v}}^n]) \in c'}$$

The membership relation is easily extended to environments and type environments:

$$\Pi \in \Gamma \Leftrightarrow \mathit{dom}(\Gamma) \subseteq \mathit{dom}(\Pi) \text{ and } \forall\, x \in \mathit{dom}(\Gamma). \Pi(x) \in \Gamma(x).$$

$$(pro)\ \frac{\forall\, i = 1..n. \vdash cd_i{:}\diamond \quad \emptyset \vdash e{:}\tau}{\vdash \overline{cd}^n\ e{:}\diamond} \qquad (cla)\ \frac{\forall\, i = 1..k.c \vdash md_i{:}\diamond \quad \mathit{fields}(c) \text{ defined}}{\vdash \textbf{class } c \textbf{ extends } c' \ \{\ \overline{fd}^n\ \overline{md}^k\ \}{:}\diamond}$$

$$(met)\ \frac{\texttt{this}{:}c, \overline{x}^n{:}\overline{\tau}^n \vdash e{:}\tau \quad \tau \leq \tau_0 \quad \mathit{override}(c, m, \overline{\tau}^n, \tau_0)}{c \vdash \tau_0\ m(\overline{\tau}^n\ \overline{x}^n)\ \{e\}{:}\diamond}$$

$$(var)\ \frac{\Gamma(x) = \tau}{\Gamma \vdash x{:}\tau} \qquad (fal)\ \frac{}{\Gamma \vdash \textbf{false}{:}\mathit{bool}} \qquad (tru)\ \frac{}{\Gamma \vdash \textbf{true}{:}\mathit{bool}}$$

$$(new)\ \frac{\forall\, i = 1..n.\Gamma \vdash e_i{:}\tau_i \quad \mathit{fields}(c) = \overline{\tau'}^n\ \overline{f}^n \quad \forall\, i = 1..n.\tau_i \leq \tau'_i}{\Gamma \vdash \textbf{new}\ c(\overline{e}^n){:}c}$$

$$(fld)\ \frac{\Gamma \vdash e{:}c \quad \mathit{fields}(c) = \overline{\tau}^n\ \overline{f}^n \quad 1 \leq i \leq n}{\Gamma \vdash e.f_i{:}\tau_i} \qquad (if)\ \frac{\Gamma \vdash e{:}\mathit{bool} \quad \Gamma \vdash e_1{:}\tau_1 \quad \Gamma \vdash e_2{:}\tau_2}{\Gamma \vdash \textbf{if}\ (e)\ e_1\ \textbf{else}\ e_2{:}\vee(\tau_1, \tau_2)}$$

$$(inv)\ \frac{\forall\, i = 0..n.\Gamma \vdash e_i{:}\tau_i \quad \mathit{meth}(\tau_0, m) = \overline{\tau'}^n\ \overline{x}^n.e{:}\tau \quad \forall\, i = 1..n.\tau_i \leq \tau'_i}{\Gamma \vdash e_0.m(\overline{e}^n){:}\tau}$$

Fig. 8. Nominal type system

Coinductive type system The coinductive type system is derived from the inductive one defined in Figure 8 as follows:

- all rules for typing expressions are interpreted coinductively (rules for well-typed programs, classes, and methods can be indifferently interpreted inductively or coinductively, since they are not recursive);
- all rules are unchanged, except for rule (inv) which is modified in (co-inv):

$$(co\text{-}inv)\ \frac{\begin{array}{l}\forall\, i = 0..n.\Gamma \Vdash e_i{:}\tau_i \quad \texttt{this}{:}\tau_0, \overline{x}^n{:}\overline{\tau}^n \Vdash e{:}\tau' \\ \mathit{meth}(\tau_0, m) = \overline{\tau'}^n\ \overline{x}^n.e{:}\tau \quad \forall\, i = 1..n.\tau_i \leq \tau'_i, \tau' \leq \tau\end{array}}{\Gamma \Vdash e_0.m(\overline{e}^n){:}\tau}$$

Rule (*co-inv*) is clearly not compositional: instead of type checking a method once for all, and using subtyping and type safe overriding (as happens in the inductive system), the coinductive type system checks a method body, not only when it is declared (rule (*cla*)), but also whenever it is called. However, from a more theoretical point of view, the coinductive type system is a step closer to the CBS. Of course the type system must be interpreted coinductively, otherwise typechecking of recursive methods would always fail. Consider for instance case 2 (b) presented in Section 4. The judgment $\emptyset \Vdash$**new** `M().m()`:`L` can be derived only with an infinite proof tree, as depicted in Figure 9. Note that the proof tree is isomorphic to the proof tree for $\emptyset \Vdash$**new** `M().m()` $\Rightarrow$ *obj*(`L`, [`n` $\mapsto$ `v`]) shown in Figure 6.

We can now prove soundness of the inductive type system w.r.t. the coinductive one.

$$\dfrac{\dfrac{}{\emptyset \Vdash \mathbf{new}\ \mathtt{M()}{:}\mathtt{M}} \qquad \dfrac{\dfrac{\dfrac{}{\mathtt{this}{:}\mathtt{M} \Vdash \mathtt{this}{:}\mathtt{M}} \quad \dfrac{\vdots}{\mathtt{this}{:}\mathtt{M} \Vdash \mathbf{new}\ \mathtt{L(this.m())}{:}\mathtt{L}}}{\mathtt{this}{:}\mathtt{M} \Vdash \mathtt{this.m()}{:}\mathtt{L}}}{\mathtt{this}{:}\mathtt{M} \Vdash \mathbf{new}\ \mathtt{L(this.m())}{:}\mathtt{L}}}{\emptyset \Vdash \mathbf{new}\ \mathtt{M().m()}{:}\mathtt{L}}$$

Fig. 9. Proof for $\emptyset \Vdash \mathbf{new}\ \mathtt{M().m()}{:}\mathtt{L}$

The following lemmas are instrumental to the proof of the theorem 5 that follows; in the claims of all lemmas we implicitly assume that judgments refer to a well-typed program. All omitted proofs can be found in the companion technical report.

Lemma 2. *If $\tau_1' \leq \tau_1$ and $\tau_2' \leq \tau_2$ then $\vee(\tau_1', \tau_2') \leq \vee(\tau_1, \tau_2)$.*

Lemma 3. *If $\mathit{fields}(c) = \overline{\tau}^n\ \overline{f}^n$, and $c' \leq c$, then $\mathit{fields}(c') = \overline{\tau}^m\ \overline{f}^m$ with $n \leq m$.*

Lemma 4. *If $\mathit{meth}(c, m) = \overline{\tau}^n\ \overline{x}^n.e{:}\tau$, then there exist c', τ' s.t. $c \leq c'$, $\tau' \leq \tau$ and $\mathtt{this}{:}c', \overline{x}^n{:}\overline{\tau}^n \vdash e{:}\tau'$.*

Lemma 5. *If $\overline{x}^n{:}\overline{\tau}^n \vdash e{:}\tau$ and for all $i = 1..n$ $\tau_i' \leq \tau_i$, then there exists τ' s.t. $\tau' \leq \tau$, and $\overline{x}^n{:}\overline{\tau'}^n \vdash e{:}\tau'$.*

Lemma 6. *The subtyping relation is transitive: if $\tau_1 \leq \tau_2$ and $\tau_2 \leq \tau_3$, then $\tau_1 \leq \tau_3$.*

Theorem 5. *Let $\overline{cd}^n$ be well-typed class declarations. If $\Gamma \vdash e{:}\tau$ in $\overline{cd}^n$, then $\Gamma \Vdash e{:}\tau$ in $\overline{cd}^n$.*

Proof. By coinduction on the rules defining the judgment $\_ \Vdash \_{:}\_$, and case analysis on e. The only interesting case is when e is a method invocation $e_0.m(\overline{e}^n)$, since for all other cases the rules of the two type systems coincide. If $\Gamma \vdash e{:}\tau$, then by definition of rule (inv) we have $\forall\, i = 0..n.\Gamma \vdash e_i{:}\tau_i$, $\mathit{meth}(\tau_0, m) = \overline{\tau'}^n\ \overline{x}^n.e{:}\tau$, and $\forall\, i = 1..n.\tau_i \leq \tau_i'$. By lemma 4 there exists τ_0', τ' s.t. $\tau_0 \leq \tau_0'$, $\tau' \leq \tau$, and $\mathtt{this}{:}\tau_0', \overline{x}^n{:}\overline{\tau'}^n \vdash e{:}\tau'$; therefore, by lemma 5 there exists τ'' s.t. $\tau'' \leq \tau'$, and hence by transitivity (lemma 6) $\tau'' \leq \tau$, and $\mathtt{this}{:}\tau_0, \overline{x}^n{:}\overline{\tau}^n \vdash e{:}\tau''$. We conclude by coinductive hypothesis and by definition of rule $(\mathit{co\text{-}inv})$.

6 Proof of soundness

In this section we prove the middle implication shown in Figure 1, which is the core of our result: soundness of the coinductive type system in terms of the CBS. Finally, by virtue of the soundness of the inductive type system w.r.t. the coinductive one (proved in Section 5), and of the soundness of the CBS w.r.t. the ISS (proved in Section 4), we can state soundness of the inductive type system in terms of the ISS as a simple corollary.

The following lemmas are instrumental to the proof of the theorem 6 that follows; in the claims of all lemmas we implicitly assume that judgments refer to a well-typed program. All omitted proofs can be found in the companion technical report.

Lemma 7. *$\tau_1 \leq \vee(\tau_1, \tau_2)$ and $\tau_2 \leq \vee(\tau_1, \tau_2)$.*

Lemma 8. *If $\mathit{fields}(c) = \overline{\tau}^n\ \overline{f}^n$, and $c \leq c'$, then $\mathit{fields}(c') = \overline{\tau}^m\ \overline{f}^m$ with $m \leq n$.*

Lemma 9 (Soundness of subtyping). *If $\mathsf{v} \in \tau$ and $\tau \leq \tau'$, then $\mathsf{v} \in \tau'$.*

Lemma 10. *If $\mathit{meth}(c, m) = \overline{\tau}^n\ \overline{x}^n.e{:}\tau$ and $c' \leq c$, then $\mathit{meth}(c', m) = \overline{\tau'}^n\ \overline{x'}^n.$ $e'{:}\tau'$ where for all $i = 1..n$ $\tau_i \leq \tau'_i$ and $\tau' \leq \tau$.*

To prove that the coinductive type system is sound w.r.t. the CBS, we coinductively define a *concretization relation* $\mathcal{R}_\gamma$ between valid proof trees ${}_{\Vdash}\nabla \in$ $\mathit{VPT}_{:}$ for $\Gamma \Vdash e{:}\tau$ and (possibly non valid) proof trees ${}_{\Rightarrow}\nabla \in \mathit{PT}_{\Rightarrow}$ for $\Pi \Vdash e \Rightarrow \mathsf{v}$, and show that for any valid proof tree ${}_{\Vdash}\nabla$ for $\Gamma \Vdash e{:}\tau$ and any $\Pi \in \Gamma$, there exists a value v and a valid proof tree ${}_{\Rightarrow}\nabla$ for $\Pi \Vdash e \Rightarrow \mathsf{v}$ s.t. ${}_{\Vdash}\nabla\ \mathcal{R}_\gamma\ {}_{\Rightarrow}\nabla$, and $\mathsf{v} \in \tau$.

Definition 9. *A relation $\mathcal{R} \subseteq \mathit{VPT}_{:} \times \mathit{PT}_{\Rightarrow}$ is a concretization iff the following constraints are satisfied:*

- $\dfrac{}{\Gamma \Vdash x{:}\tau}\ \mathcal{R}\ \dfrac{}{\Pi \Vdash x \Rightarrow \mathsf{v}}$ *iff $\Pi \in \Gamma$, and $\mathit{ok}(\textsc{var})\left(\dfrac{}{\Pi \Vdash x \Rightarrow \mathsf{v}}\right)$*
- $\dfrac{}{\Gamma \Vdash \mathbf{false}{:}\mathit{bool}}\ \mathcal{R}\ \dfrac{}{\Pi \Vdash \mathbf{false} \Rightarrow \mathit{false}}$ *iff $\Pi \in \Gamma$*
- $\dfrac{}{\Gamma \Vdash \mathbf{true}{:}\mathit{bool}}\ \mathcal{R}\ \dfrac{}{\Pi \Vdash \mathbf{true} \Rightarrow \mathit{true}}$ *iff $\Pi \in \Gamma$*
- $\dfrac{\overline{{}_{\Vdash}\nabla}^n}{\Gamma \Vdash \mathbf{new}\ c(\overline{e}^n){:}c}\ \mathcal{R}\ \dfrac{\overline{{}_{\Rightarrow}\nabla}^n}{\Pi \Vdash \mathbf{new}\ c(\overline{e}^n) \Rightarrow \mathsf{v}}$ *iff for all $i = 1..n$ ${}_{\Vdash}\nabla_i\ \mathcal{R}\ {}_{\Rightarrow}\nabla_i$, $\Pi \in \Gamma$, and* $\mathit{ok}(\textsc{new})\left(\dfrac{\overline{{}_{\Rightarrow}\nabla}^n}{\Pi \Vdash \mathbf{new}\ c(\overline{e}^n) \Rightarrow \mathsf{v}}\right)$
- $\dfrac{{}_{\Vdash}\nabla}{\Gamma \Vdash e.f{:}\tau}\ \mathcal{R}\ \dfrac{{}_{\Rightarrow}\nabla}{\Pi \Vdash e.f \Rightarrow \mathsf{v}}$ *iff ${}_{\Vdash}\nabla\ \mathcal{R}\ {}_{\Rightarrow}\nabla$, $\mathsf{v} \in \tau$, $\Pi \in \Gamma$, and $\mathit{ok}(\textsc{fld})\left(\dfrac{{}_{\Rightarrow}\nabla}{\Pi \Vdash e.f \Rightarrow \mathsf{v}}\right)$*
- $\dfrac{{}_{\Vdash}\nabla\ \ {}_{\Vdash}\nabla_1\ \ {}_{\Vdash}\nabla_2}{\Gamma \Vdash \mathbf{if}\ (e)\ e_1\ \mathbf{else}\ e_2{:}\tau}\ \mathcal{R}\ \dfrac{{}_{\Rightarrow}\nabla\ \ {}_{\Rightarrow}\nabla_1}{\Pi \Vdash \mathbf{if}\ (e)\ e_1\ \mathbf{else}\ e_2 \Rightarrow \mathsf{v}}$ *iff ${}_{\Vdash}\nabla\ \mathcal{R}\ {}_{\Rightarrow}\nabla$, ${}_{\Vdash}\nabla_1\ \mathcal{R}\ {}_{\Rightarrow}\nabla_1$,* *$\mathit{root}({}_{\Rightarrow}\nabla) = \Pi \Vdash e \Rightarrow \mathit{true}$, $\mathsf{v} \in \tau$, $\Pi \in \Gamma$, and $\mathit{ok}(\textsc{ift})\left(\dfrac{{}_{\Rightarrow}\nabla\ \ {}_{\Rightarrow}\nabla_1}{\Pi \Vdash \mathbf{if}\ (e)\ e_1\ \mathbf{else}\ e_2 \Rightarrow \mathsf{v}}\right)$*

– $\dfrac{{}_{\Vdash}\nabla\ \ {}_{\Vdash}\nabla_1\ \ {}_{\Vdash}\nabla_2}{\Gamma \Vdash \textbf{if}\ (e)\ e_1\ \textbf{else}\ e_2{:}\tau}\ \mathcal{R}\ \dfrac{{}_{\Rightarrow}\nabla\ \ {}_{\Rightarrow}\nabla_2}{\Pi \Vdash \textbf{if}\ (e)\ e_1\ \textbf{else}\ e_2 \Rightarrow \mathtt{v}}$ *iff* ${}_{\Vdash}\nabla\ \mathcal{R}\ {}_{\Rightarrow}\nabla,\ {}_{\Vdash}\nabla_2\ \mathcal{R}\ {}_{\Rightarrow}\nabla_2,$ $root({}_{\Rightarrow}\nabla) = \Pi \Vdash e \Rightarrow false,\ \mathtt{v} \in \tau,\ \Pi \in \Gamma,$ *and* $ok(\textsc{iff})\left(\dfrac{{}_{\Rightarrow}\nabla\ \ {}_{\Rightarrow}\nabla_2}{\Pi \Vdash \textbf{if}\ (e)\ e_1\ \textbf{else}\ e_2 \Rightarrow \mathtt{v}}\right)$

– $\dfrac{{}_{\Vdash}\nabla_0\ \ \overline{{}_{\Vdash}\nabla}^n\ \ {}_{\Vdash}\nabla}{\Gamma \Vdash e_0.m(\overline{e}^n){:}\tau}\ \mathcal{R}\ \dfrac{{}_{\Rightarrow}\nabla_0\ \ \overline{{}_{\Rightarrow}\nabla}^n\ \ {}_{\Rightarrow}\nabla}{\Pi \Vdash e_0.m(\overline{e}^n) \Rightarrow \mathtt{v}}$ *iff for all* $i = 0..n\ {}_{\Vdash}\nabla_i \mathcal{R}\, {}_{\Rightarrow}\nabla_i,\ {}_{\Vdash}\nabla \mathcal{R}\, {}_{\Rightarrow}\nabla,\ \mathtt{v} \in \tau,$ $\Pi \in \Gamma,$ *and* $ok(\textsc{inv})\left(\dfrac{{}_{\Rightarrow}\nabla_0\ \ \overline{{}_{\Rightarrow}\nabla}^n\ \ {}_{\Rightarrow}\nabla}{\Pi \Vdash e_0.m(\overline{e}^n) \Rightarrow \mathtt{v}}\right).$

The function $\mathcal{F}$ corresponding to the recursive definition of concretization relation is trivially monotone on the complete lattice defined by the power set of $VPT_{:} \times PT_{\Rightarrow}$, therefore by the Tarski-Knaster theorem there exists the greatest concretization relation, denoted by $\mathcal{R}_\gamma$. However, the Tarski-Knaster theorem does not provide any guarantee that for any valid proof tree ${}_{\Vdash}\nabla$ for $\Gamma \Vdash e{:}\tau$ and any $\Pi \in \Gamma$, there exists a value $\mathtt{v}$ and a valid proof tree ${}_{\Rightarrow}\nabla$ for $\Pi \Vdash e \Rightarrow \mathtt{v}$ s.t. ${}_{\Vdash}\nabla\ \mathcal{R}_\gamma\ {}_{\Rightarrow}\nabla$, and $\mathtt{v} \in \tau$. To prove such a property we need to apply the Kleene theorem; indeed, $\mathcal{F}$ also preserves infima of descending chains in the same complete lattice, hence, the concretization relation $\mathcal{R}_\gamma$ is defined by $\inf\{\mathcal{F}^n(\top) \mid n \in \mathbb{N}\}$, where $\top$ denotes the top element of the lattice defined by the power set of $VPT_{:} \times PT_{\Rightarrow}$, that is, the relation associating any valid proof tree for the judgment $\_ \Vdash \_{:}\_$, with any proof tree for the judgment $\_ \Vdash \_ \Rightarrow \_$. We abbreviate $\mathcal{F}^n(\top)$ with $\mathcal{R}^n_\gamma$, hence $\mathcal{R}^0_\gamma = \top$.

As an example, we have that ${}_{\Vdash}\nabla^e \mathcal{R}_\gamma {}_{\Rightarrow}\nabla^e$, where ${}_{\Vdash}\nabla^e$ and ${}_{\Rightarrow}\nabla^e$ are the proof trees defined in Figure 9 and 6, respectively. The easiest way to prove this fact is to show that there exists a concretization relation $\mathcal{R}$ s.t. ${}_{\Vdash}\nabla^e\ \mathcal{R}\ {}_{\Rightarrow}\nabla^e$; this can be achieved by considering the finite relation that associates each subtree of ${}_{\Vdash}\nabla^e$ (including ${}_{\Vdash}\nabla^e$ itself), with the corresponding subtree[3] of ${}_{\Rightarrow}\nabla^e$ (including ${}_{\Rightarrow}\nabla^e$ itself); it is immediate to verify that such a relation is a concretization.

However, when proving soundness the proof tree for the CBS is unknown, and therefore its existence is proved by showing that it can be obtained as the limit of a Cauchy sequence in a complete metric space. Therefore, to better understand the proof that will follow, it is instructive to show how the proof tree ${}_{\Rightarrow}\nabla^e$ can actually be built from ${}_{\Vdash}\nabla^e$ by using the Kleene construction. We assume that the expression is evaluated in a program where the only available classes are `M` and `L` as declared for case 2 (b) in Section 4, and we use ellipses ... as a wildcard.

– ${}_{\Vdash}\nabla^e \mathcal{R}^0_\gamma {}_{\Rightarrow}\nabla$ for any ${}_{\Rightarrow}\nabla \in PT_{\Rightarrow}$.
– ${}_{\Vdash}\nabla^e \mathcal{R}^1_\gamma {}_{\Rightarrow}\nabla$ for any ${}_{\Rightarrow}\nabla \in PT_{\Rightarrow}$ s.t. ${}_{\Rightarrow}\nabla =$

[3] We recall that the two trees are isomorphic; furthermore, they have a finite number of subtrees, since they are regular.

$$\dfrac{\dfrac{\ldots}{\Pi \Vdash \textbf{new}\ \texttt{M()} \Rightarrow \mathtt{v}} \quad \dfrac{\ldots}{\texttt{this} \mapsto \mathtt{v} \Vdash \textbf{new}\ \texttt{L(this.m())} \Rightarrow \mathtt{v}_0}}{\Pi \Vdash \textbf{new}\ \texttt{M().m()} \Rightarrow \mathtt{v}_0}$$

where $\Pi \in \emptyset$ (hence, Π can be any value environment), $\mathtt{v}$ can be any value, and $\mathtt{v}_0 \in \mathtt{L}$, hence for all $i \in \mathbb{N}$, $\mathtt{v}_i = obj(\mathtt{L}, [\mathtt{n} \mapsto \mathtt{v}_{i+1}])$, therefore $\mathtt{v}_0$ is the unique value s.t. $\mathtt{v}_0 = obj(\mathtt{L}, [\mathtt{n} \mapsto \mathtt{v}_0])$ and $\mathtt{v}_i = \mathtt{v}_{i+1}$, for all $i \in \mathbb{N}$. Note that, since we have assumed that `M` and `L` are the only available classes of the program, there exists only one possible subtype of `L`, namely `L` itself, and the equations above can be directly derived by applying membership rule OBJ. Therefore, for this particular case we get the returned value (in this particular case it is unique) just at the first iteration; however, for getting the corresponding valid proof tree all iterations have to be considered.
We proceed with the next iteration, to show how at each step the obtained proof trees are better approximations of a valid proof tree.

– ${}_{\Vdash}\nabla^e \mathcal{R}^2_{\gamma} {}_{\Rightarrow}\nabla$ for any ${}_{\Rightarrow}\nabla \in PT_{\Rightarrow}$ s.t. ${}_{\Rightarrow}\nabla =$

$$\dfrac{\dfrac{}{\Pi \Vdash \textbf{new}\ \texttt{M()} \Rightarrow obj(\texttt{M}, [\,])} \quad \dfrac{\dfrac{\ldots}{\texttt{this} \mapsto obj(\texttt{M}, [\,]) \Vdash \texttt{this.m()} \Rightarrow \mathtt{v}_0}}{\texttt{this} \mapsto obj(\texttt{M}, [\,]) \Vdash \textbf{new}\ \texttt{L(this.m())} \Rightarrow \mathtt{v}_0}}{\Pi \Vdash \textbf{new}\ \texttt{M().m()} \Rightarrow \mathtt{v}_0}$$

where $\Pi \in \emptyset$ (hence, Π can be any value environment). Note that, by virtue of the equation $\mathtt{v}_0 = obj(\mathtt{L}, [\mathtt{n} \mapsto \mathtt{v}_0])$, the evaluation of **new** `L(this.m())` and of `this.m()` returns the same vale $\mathtt{v}_0$.

It can be easily proved by standard induction over n that ${}_{\Vdash}\nabla^e \mathcal{R}^n_{\gamma} {}_{\Rightarrow}\nabla^e$, for all $n \in \mathbb{N}$, where ${}_{\Rightarrow}\nabla^e$ is the valid proof tree defined in Figure 6; since $\mathcal{R}_\gamma$ is the greatest lower bound of $\{\mathcal{R}^n_\gamma | n \in \mathbb{N}\}$, we obtain that ${}_{\Vdash}\nabla^e \mathcal{R}_{\gamma} {}_{\Rightarrow}\nabla^e$.

To prove the main claim from which soundness of the coinductive type system w.r.t. the CBS can be derived, we need to define a complete metric space of proof trees for the CBS.

We first define the metric of value environment. We recall that a value environment is a finite partial function mapping variables to values, and that values are finitely branching trees with infinite paths (hence, they form a complete metric space with the distance d_T of Definition 1).

Proposition 2. *The set of value environments forms a complete metric space when equipped with the distance d_Π defined as follows:*

$$d_\Pi(\Pi_1, \Pi_2) = \begin{cases} 1 & \text{if } dom(\Pi_1) \neq dom(\Pi_2) \\ \max(\{0, d_T(\Pi_1(x), \Pi_2(x)) \mid x \in D\}) & \text{if } D = dom(\Pi_1) = dom(\Pi_2) \end{cases}$$

Proof. See the companion technical report.

Proposition 3. *The set of pairs of value environments and values forms a complete metric space when equipped with the distance $d_{\Pi,\mathtt{v}}$ defined as follows:*

$$d_{\Pi,\mathtt{v}}((\Pi_1, \mathtt{v}_1), (\Pi_2, \mathtt{v}_2)) = \max\{d_\Pi(\Pi_1, \Pi_2), d_T(\mathtt{v}_1, \mathtt{v}_2)\}$$

Proof. A well known property of product metric spaces that can be easily checked. From Proposition 2 one can easily deduce that $0 \leq d_{\Pi,\mathtt{v}}((\Pi_1, \mathtt{v}_1), (\Pi_2, \mathtt{v}_2)) \leq 1$, since $d_{\Pi,\mathtt{v}}((\Pi_1, \mathtt{v}_1), (\Pi_2, \mathtt{v}_2)) \in \{0\} \cup \{2^{-c} \mid c \in \mathbb{N}\}$.

Let j be the judgment $\Pi \Vdash e \Rightarrow \mathtt{v}$, then $ev(j)$ and $exp(j)$ denote $(\Pi, \mathtt{v})$ and e, respectively; furthermore, $exp({}_{\Rightarrow}\nabla)$ denotes the tree t over expressions s.t. $dom(t) = dom({}_{\Rightarrow}\nabla)$, and for all $p \in dom(t)$ $t(p) = exp({}_{\Rightarrow}\nabla(p))$.

Proposition 4. *The set $PT_{\Rightarrow}$ of proof trees for $\Pi \Vdash e \Rightarrow \mathtt{v}$ forms a complete metric space when equipped with the distance d_∇ defined as follows:*

$$d_\nabla({}_{\Rightarrow}\nabla_1, {}_{\Rightarrow}\nabla_2) = \max(\{2^{-c}\} \cup S) \textit{ where}$$

$S = \{2^{-k} \cdot d_{\Pi,\mathtt{v}}(ev({}_{\Rightarrow}\nabla_1(p)), ev({}_{\Rightarrow}\nabla_2(p))) \mid p \in \mathbb{N}^k \cap dom({}_{\Rightarrow}\nabla_1), 0 \leq k < c\}$
$c = shtp(exp({}_{\Rightarrow}\nabla_1), exp({}_{\Rightarrow}\nabla_2))$*, that is,* $c = \min\{n \in \mathbb{N} \mid p \in \mathbb{N}^n, exp({}_{\Rightarrow}\nabla_1(p)) \neq_\perp exp({}_{\Rightarrow}\nabla_2(p))\}$ *(see Proposition 1 for the definition of shtp and the related notation).*

Proof. See the companion technical report.

Let us consider the Kleene approximations $\mathcal{R}^i_\gamma$ $(i \in \mathbb{N})$ of the concretization relation $\mathcal{R}_\gamma$. Then the following lemma holds, where we assume that judgments are indexed over a class table corresponding to a sequence of well-typed classes $\overline{cd}^n$.

Lemma 11 (Substitution). *Let ${}_{\Vdash}\nabla$ be a valid proof tree for $\Gamma \Vdash e{:}\tau$, and ${}_{\Rightarrow}\nabla$ a (not necessarily valid) proof tree for $\Pi \Vdash e \Rightarrow \mathtt{v}$. For all $n \in \mathbb{N}$, if the following facts hold:*

1. ${}_{\Vdash}\nabla \mathcal{R}^n_\gamma {}_{\Rightarrow}\nabla$
2. $\Pi, \Pi' \in \Gamma$, $d_\Pi(\Pi, \Pi') \leq 2^{-n}$
3. *there exists* ${}_{\Rightarrow}\nabla'$ *s.t.* ${}_{\Vdash}\nabla \mathcal{R}^{n+1}_\gamma {}_{\Rightarrow}\nabla'$ *and* $d_\nabla({}_{\Rightarrow}\nabla, {}_{\Rightarrow}\nabla') \leq 2^{-n}$

then there exists a proof tree ${}_{\Rightarrow}\nabla''$ for $\Pi' \Vdash e \Rightarrow \mathtt{v}'$ s.t. ${}_{\Vdash}\nabla \mathcal{R}^{n+1}_\gamma {}_{\Rightarrow}\nabla''$ and $d_\nabla({}_{\Rightarrow}\nabla, {}_{\Rightarrow}\nabla'') \leq 2^{-n}$.

Proof. The proof is by induction on n, and by case analysis on the expression e.

Lemma 12. *For all $n \in \mathbb{N}$, ${}_{\Vdash}\nabla \in VPT_{:}$, and ${}_{\Rightarrow}\nabla \in PT_{\Rightarrow}$, if ${}_{\Vdash}\nabla \mathcal{R}^n_\gamma {}_{\Rightarrow}\nabla$, then there exists ${}_{\Rightarrow}\nabla'$ s.t. ${}_{\Vdash}\nabla \mathcal{R}^{n+1}_\gamma {}_{\Rightarrow}\nabla'$, and $d_\nabla({}_{\Rightarrow}\nabla, {}_{\Rightarrow}\nabla') \leq 2^{-n}$.*

Proof. The proof is by induction on n, and by case analysis on the expression e.

Basis: If $n = 0$, then by definition $\mathcal{R}^0_\gamma = \top$, and, hence, ${}_{\Vdash}\nabla \mathcal{R}^0_\gamma {}_{\Rightarrow}\nabla$ for all ${}_{\Vdash}\nabla \in VPT_{:}$, and ${}_{\Rightarrow}\nabla \in PT_{\Rightarrow}$; therefore we have to show that there exists ${}_{\Rightarrow}\nabla'$ s.t. ${}_{\Vdash}\nabla \mathcal{R}^1_\gamma {}_{\Rightarrow}\nabla'$. Let us consider the case where $e = e_0.m(\overline{e}^n)$ (for all other cases the proof is analogous). If ${}_{\Vdash}\nabla$ is a proof tree for $\Gamma \Vdash e_0.m(\overline{e}^n){:}\tau$, then by rule (*co-inv*) we have that $\Gamma \Vdash e_0{:}\tau_0$ and $meth(\tau_0, m) = \overline{\tau'}^n\ \overline{x}^n.e{:}\tau$. By definition of membership, there always exist Π and $\mathtt{v}$ s.t. $\Pi \in \Gamma$, and $\mathtt{v} \in \tau$, hence we can pick any Π and $\mathtt{v}$ s.t. $\Pi \in \Gamma$, and $\mathtt{v} \in \tau$, and build the following (not necessarily valid) proof tree:

$$
{}_{\Rightarrow}\nabla' = \dfrac{\forall\, i = 0..n. \dfrac{}{\Pi \Vdash e_i \Rightarrow \mathtt{v}_i} \qquad \dfrac{}{\mathtt{this} \mapsto \mathtt{v}_0, \overline{x}^n \mapsto \overline{\mathtt{v}}^n \Vdash e \Rightarrow \mathtt{v}}}{\Pi \Vdash e_0.m(\overline{e}^n) \Rightarrow \mathtt{v}}
$$

with $\mathtt{v}_0 = \mathit{obj}(\tau_0, [\ldots])$ and $\mathit{meth}(\tau_0, m) = \overline{\tau'}^n\ \overline{x}^n.e{:}\tau$. Clearly, ${}_{\Vdash}\nabla \mathcal{R}^1_\gamma {}_{\Rightarrow}\nabla'$, since by definition 9, and by definition of $\mathcal{R}^1_\gamma$,

$\dfrac{{}_{\Vdash}\nabla_0\ \overline{{}_{\Vdash}\nabla}^n\ {}_{\Vdash}\nabla}{\Gamma \Vdash e_0.m(\overline{e}^n){:}\tau}\ \mathcal{R}^1_\gamma\ \dfrac{{}_{\Rightarrow}\nabla_0\ \overline{{}_{\Rightarrow}\nabla}^n\ {}_{\Rightarrow}\nabla}{\Pi \Vdash e_0.m(\overline{e}^n) \Rightarrow \mathtt{v}}$ iff for all $i = 0..n$ ${}_{\Vdash}\nabla_i \mathcal{R}^0_\gamma {}_{\Rightarrow}\nabla_i$, ${}_{\Vdash}\nabla \mathcal{R}^0_\gamma {}_{\Rightarrow}\nabla$, $\mathtt{v} \in \tau$, $\Pi \in \Gamma$, and $\mathit{ok}(\textsc{inv})\left(\dfrac{{}_{\Rightarrow}\nabla_0\ \overline{{}_{\Rightarrow}\nabla}^n\ {}_{\Rightarrow}\nabla}{\Pi \Vdash e_0.m(\overline{e}^n) \Rightarrow \mathtt{v}}\right)$.

Finally, by Proposition 4 we have that $d_\nabla({}_{\Rightarrow}\nabla, {}_{\Rightarrow}\nabla') \leq 2^{-0} = 1$ for all ${}_{\Rightarrow}\nabla, {}_{\Rightarrow}\nabla' \in PT_{\Rightarrow}$.

Inductive step: we have to prove that for all $n \geq 1$, ${}_{\Vdash}\nabla \mathcal{R}^{n-1}_\gamma {}_{\Rightarrow}\nabla \Rightarrow \exists\, {}_{\Rightarrow}\nabla'$ s.t. ${}_{\Vdash}\nabla \mathcal{R}^n_\gamma {}_{\Rightarrow}\nabla'$, and $d_\nabla({}_{\Rightarrow}\nabla, {}_{\Rightarrow}\nabla') \leq 2^{-n+1}$ implies ${}_{\Vdash}\nabla \mathcal{R}^n_\gamma {}_{\Rightarrow}\nabla \Rightarrow \exists\, {}_{\Rightarrow}\nabla'$ s.t. ${}_{\Vdash}\nabla \mathcal{R}^{n+1}_\gamma {}_{\Rightarrow}\nabla'$, and $d_\nabla({}_{\Rightarrow}\nabla, {}_{\Rightarrow}\nabla') \leq 2^{-n}$.

As for the basis, we consider the case where $e = e_0.m(\overline{e}^n)$ (for all other cases the proof is analogous). Therefore let us assume that ${}_{\Vdash}\nabla \mathcal{R}^n_\gamma {}_{\Rightarrow}\nabla$, where ${}_{\Vdash}\nabla$ is a valid proof tree for $\Gamma \Vdash e_0.m(\overline{e}^n){:}\tau$. By rule (*co-inv*) we have

$$
{}_{\Vdash}\nabla = \dfrac{{}_{\Vdash}\nabla_0\ \overline{{}_{\Vdash}\nabla}^n\ {}_{\Vdash}\nabla'}{\Gamma \Vdash e_0.m(\overline{e}^n){:}\tau}
$$

with $\mathit{meth}(\tau_0, m) = \overline{\tau'}^n\ \overline{x}^n.e{:}\tau$, $\forall\, i = 1..n.\tau_i \leq \tau'_i$, $\tau' \leq \tau$, and where $\forall\, i = 1..n.\mathit{root}({}_{\Vdash}\nabla_i) = \dfrac{\vdots}{\Gamma \Vdash e_i{:}\tau_i}$, $\mathit{root}({}_{\Vdash}\nabla') = \dfrac{\vdots}{\mathtt{this}{:}\tau_0, \overline{x}^n{:}\overline{\tau}^n \Vdash e{:}\tau'}$.

Since ${}_{\Vdash}\nabla \mathcal{R}^n_\gamma {}_{\Rightarrow}\nabla$, by Definition 9 and by definition of $\mathcal{R}^n_\gamma$ we have

$$
{}_{\Rightarrow}\nabla = \dfrac{{}_{\Rightarrow}\nabla_0\ \overline{{}_{\Rightarrow}\nabla}^n\ {}_{\Rightarrow}\nabla'}{\Pi \Vdash e_0.m(\overline{e}^n) \Rightarrow \mathtt{v}}
$$

and for all $i = 0..n$ ${}_{\Vdash}\nabla_i \mathcal{R}^{n-1}_\gamma {}_{\Rightarrow}\nabla_i$, ${}_{\Vdash}\nabla' \mathcal{R}^{n-1}_\gamma {}_{\Rightarrow}\nabla'$, $\mathtt{v} \in \tau$, $\Pi \in \Gamma$, and $\mathit{ok}(\textsc{inv})({}_{\Rightarrow}\nabla)$. Then by inductive hypothesis we have that there exist ${}_{\Rightarrow}\nabla'_0, \ldots, {}_{\Rightarrow}\nabla'_n$ and ${}_{\Rightarrow}\nabla''$ s.t. for all $i = 0..n$ ${}_{\Vdash}\nabla_i \mathcal{R}^n_\gamma {}_{\Rightarrow}\nabla'_i$, $d_\nabla({}_{\Rightarrow}\nabla_i, {}_{\Rightarrow}\nabla'_i) \leq 2^{-n+1}$, ${}_{\Vdash}\nabla' \mathcal{R}^n_\gamma {}_{\Rightarrow}\nabla''$, $d_\nabla({}_{\Rightarrow}\nabla', {}_{\Rightarrow}\nabla'') \leq 2^{-n+1}$. Therefore we have that for all $i = 0..n$ $\mathit{root}({}_{\Rightarrow}\nabla'_i) = \Pi_i \Vdash e_i \Rightarrow \mathtt{v}_i$, $\mathit{root}({}_{\Rightarrow}\nabla'') = \Pi' \Vdash e \Rightarrow \mathtt{v}'$, with $\Pi_i \in \Gamma$, $\Pi' \in (\mathtt{this}{:}\tau_0, \overline{x}^n{:}\overline{\tau}^n)$, $\mathtt{v}_i \in \tau_i$ (hence, $\mathtt{v}_0 = \mathit{obj}(\tau_0, [\ldots])$) and $\mathtt{v}' \in \tau$. By lemma 11 we can derive from ${}_{\Rightarrow}\nabla'_i$ ($i = 0..n$) and from ${}_{\Rightarrow}\nabla''$ the proof trees ${}_{\Rightarrow}\nabla''_i$ ($i = 0..n$) and ${}_{\Rightarrow}\nabla'''$ s.t. for all $i = 0..n$ ${}_{\Vdash}\nabla_i \mathcal{R}^n_\gamma {}_{\Rightarrow}\nabla''_i$,

$d_\nabla({}_{\Rightarrow}\nabla_i, {}_{\Rightarrow}\nabla''_i) \le 2^{-n+1}$, ${}_{\Vdash}\nabla' \mathcal{R}^n_\gamma {}_{\Rightarrow}\nabla'''$, $d_\nabla({}_{\Rightarrow}\nabla', {}_{\Rightarrow}\nabla''') \le 2^{-n+1}$, and $root({}_{\Rightarrow}\nabla''_i) = \Pi \Vdash e_i \Rightarrow \mathtt{v}'_i$, $root({}_{\Rightarrow}\nabla''') = \mathtt{this} \mapsto \mathtt{v}'_0, \overline{x}^n \mapsto \overline{\mathtt{v}'}^n \Vdash e \Rightarrow \mathtt{v}''$

Finally, the proof tree

$$
{}_{\Vdash}\bar{\nabla} = \frac{{}_{\Vdash}\nabla''_0 \ \ \overline{{}_{\Vdash}\nabla''}^n \ \ {}_{\Vdash}\nabla'''}{\Gamma \Vdash e_0.m(\overline{e}^n):\tau}
$$

is s.t. ${}_{\Vdash}\nabla \mathcal{R}^{n+1}_\gamma {}_{\Rightarrow}\bar{\nabla}$, and $d_\nabla({}_{\Rightarrow}\nabla, {}_{\Rightarrow}\bar{\nabla}) \le 2^{-n}$, by definition of $\mathcal{R}^{n+1}_\gamma$ and d_∇.

We can now state the main result.

Theorem 6. *Let $\overline{cd}^n$ be well-typed class declarations. If $\Gamma \Vdash e{:}\tau$ and $\Pi \in \Gamma$ in $\overline{cd}^n$, then there exists $\mathtt{v}$ s.t. $\Pi \Vdash e \Rightarrow \mathtt{v}$ and $\mathtt{v} \in \tau$ in $\overline{cd}^n$.*

Proof. Let ${}_{\Vdash}\nabla$ be a proof tree for $\Gamma \Vdash e{:}\tau$; directly from lemma 12 we deduce that it is possible to build a Cauchy sequence $({}_{\Rightarrow}\nabla_i)_{i\in\mathbb{N}}$ of proof trees s.t. ${}_{\Vdash}\nabla \mathcal{R}^i_\gamma {}_{\Rightarrow}\nabla_i$ for all $i \in \mathbb{N}$; by Proposition 4, such a sequence has a certain limit ${}_{\Rightarrow}\nabla$, s.t. ${}_{\Vdash}\nabla \mathcal{R}_\gamma {}_{\Rightarrow}\nabla$, which is a valid proof tree for $\Pi \Vdash e \Rightarrow \mathtt{v}$, with $\mathtt{v} \in \tau$. Note that, if the metric space of proof trees is not complete, then we could not deduce that the sequence $({}_{\Rightarrow}\nabla_i)_{i\in\mathbb{N}}$ has a limit; indeed, if we restrict the CBS to finite or regular values, then it is not possible to define a complete metric space, and, therefore, the sequence $({}_{\Rightarrow}\nabla_i)_{i\in\mathbb{N}}$ has no limit, and the claim of soundness does not hold, as already observed in the examples 2 (b) and (c) in Section 4.

Soundness of the inductive type system in terms of the CBS and of the ISS can be derived as two simple corollaries.

Corollary 2. *Let $\overline{cd}^n$ be well-typed class declarations. If $\Gamma \vdash e{:}\tau$, and $\Pi \in \Gamma$ in $\overline{cd}^n$, then there exists $\mathtt{v}$ s.t. $\Pi \Vdash e \Rightarrow \mathtt{v}$ and $\mathtt{v} \in \tau$ in $\overline{cd}^n$.*

Proof. The theorem is a straightforward corollary of Theorems 5 and 6.

Corollary 3. *If $\emptyset \vdash e{:}\tau$, $e \xrightarrow{*} e'$, and e' is a normal form, then e' is a value.*

Proof. Direct from corollaries 2 and 1.

Such a corollary is sufficient for guaranteeing the soundness of the type system in terms of the ISS: a well-typed expression can never get stuck in the ISS. However, by adding the following property (that can be proved easily), we can also deduce that the value e' is s.t. $\emptyset \vdash e'{:}\tau'$ with $\tau' \le \tau$.

Proposition 5. *If $\emptyset \Vdash v \Rightarrow \mathtt{v}$, and $\mathtt{v} \in \tau$, then $\emptyset \vdash v{:}\tau'$, with $\tau' \le \tau$.*

We can now prove the generalization of Corollary 3.

Corollary 4. *If $\emptyset \vdash e{:}\tau$, $e \xrightarrow{*} e'$, and e' is a normal form, then e' is a value and $\emptyset \vdash e'{:}\tau'$ with $\tau' \le \tau$.*

Proof. By Corollary 2 we know also that $\mathtt{v} \in \tau$, and by Corollary 1 we know that $\emptyset \Vdash e' \Rightarrow \mathtt{v}$, hence we can conclude by Proposition 5.

We end this section by providing a generic scheme to be adopted for proving soundness of a type system in terms of the CBS of a language. We consider the case where one would like also to relate the CBS to the ISS, and derive from such a relation a standard soundness claim expressed in terms of the ISS.

We assume that the ISS is defined by a reduction relation $e_1 \rightarrow e_2$, and a set of values v (which are a subset of expressions in normal form), and the CBS is defined by a judgment $\Pi \Vdash e \Rightarrow \mathtt{v}$, where Π is an environment associating variables with values, and $\mathtt{v}$ is a value (all definitions are expected to be coinductive). The type system is defined by a judgment $\Gamma \vdash e{:}\tau$, where Γ is a type environment associating variables with types, and τ is a type, and subtyping $\tau_1 \leq \tau_2$ and membership $\mathtt{v} \in \tau$ (which is easily extended to environments) are defined. Finally, a coinductive type system, defined by a judgment $\Gamma \Vdash e{:}\tau$, can be routinely defined from the inductive one.

Then the following properties have to be proved:

1. If $\emptyset \Vdash e \Rightarrow \mathtt{v}$, then either e is a value, or there exists e' s.t. $e \rightarrow e'$.
2. If $\emptyset \Vdash e \Rightarrow \mathtt{v}$, and $e \rightarrow e'$, then $\emptyset \Vdash e' \Rightarrow \mathtt{v}$.
3. If $\Gamma \vdash e{:}\tau$, then $\Gamma \Vdash e{:}\tau$.
4. If $\Gamma \Vdash e{:}\tau$, and $\Pi \in \Gamma$, then there exists $\mathtt{v}$ s.t. $\Pi \Vdash e \Rightarrow \mathtt{v}$ and $\mathtt{v} \in \tau$.
5. If $\emptyset \Vdash v \Rightarrow \mathtt{v}$, and $\mathtt{v} \in \tau$, then $\emptyset \vdash v{:}\tau'$, with $\tau' \leq \tau$.

We stress again that primitive properties 1 and 2 involve ISS and CBS only and, hence, can be proved once per each language, and reused for any type system.

From the claim above one can derive the following properties:

– If $\emptyset \vdash e{:}\tau$, $e \xrightarrow{*} e'$, and e' is a normal form, then e' is a value. Derivable from claims 1,2, 3, and 4.
– If $\emptyset \vdash e{:}\tau$, $e \xrightarrow{*} e'$, and e' is a normal form, then e' is a value and $\emptyset \vdash e{:}\tau'$ with $\tau' \leq \tau$. Derivable if claim 5 holds as well.

7 Conclusion and related work

We have shown that it is possible to prove soundness of a conventional inductive and nominal type system for a Java-like language in terms of a coinductive big-step operational semantics obtained by interpreting coinductively the rules of the standard big-step semantics. We have also suggested a generic scheme, where parts of the proofs can be reused, to be adopted for proving soundness of a type system in terms of the CBS of a language. The key point of the result is that infinite (including non regular) values have to be considered, otherwise the claim fails. Infinite values allow the definition of a complete metric space of proof trees for the CBS, which ensures that every well-typed expression evaluates into a value in the CBS, even in case of non-termination.

We have also shown that the CBS can be regarded as the concretization of a coinductive type system that can be directly derived from the standard inductive type system. Beside making the proof of soundness clearer, this fact also reveals how coinduction is related to the inductive type system.

With respect to the traditional one, the proposed approach may seem overly more complex, although big-step operational semantics tend to be simpler than small-step ones, especially when one wants to model more significant subsets of a real language. The main source of complexity comes from the proofs in Section 6, and from the fact that coinduction is less intuitive than induction. It would be worth investigating whether coalgebraic techniques could be used, to avoid using complete metric spaces. However, we hope that the provided definitions and proofs can be easily adapted for other type systems and languages.

The pioneering work of Milner and Tofte [14] is one of the first where coinduction is used for proving consistency of the type system and the big-step semantics of a simple functional language; however rules are interpreted inductively, and the semantics does not capture diverging evaluations.

In their work Leroy and Grall [13] analyze two kinds of coinductive big-step operational semantics for the call-by-value λ-calculus, study their relationships with the small-step and denotational semantics, and their suitability for compiler correctness proofs. Besides the fact that here we consider a Java-like language, the main contribution of this paper w.r.t. Leroy and Grall's work is showing that by interpreting coinductively a standard big-step operational semantics, soundness of a standard nominal type system can be proved. We could prove such a result because (1) in our semantics not only evaluation rules are interpreted coinductively, but also the definition of values, and (2) the absence of first-class functions in our language makes the treatment simpler. Leroy and Grall show that a similar soundness claim does not hold in their setting; we conjecture that the only reason for that consists in the fact that in their coinductive semantics values are defined inductively (hence are finite), rather than coinductively (that is, infinite). It would be interesting to investigate whether soundness holds for the λ-calculus when values are defined coinductively.

Kusmierek and Bono propose a different approach and prove type soundness w.r.t. an inductive big-step operational semantics; their proposal is centered on the idea of tracing the intermediate steps of a program execution with a partial derivation-search algorithm which deterministically computes the value and the proof tree of evaluation judgments. Similar approaches, although their corresponding semantics are not deterministic, are those of Ager [1] and Stoughton [18].

Nakata and Uustalu [16,15] define a coinductive trace-based semantics, whose main aim, however, is formal verification of non-terminating programs.

Finally we would like to mention the work by Ernst et al. [10] where a soundness result w.r.t. a big-step operational semantics is proved thanks to a coverage lemma ensuring that errors do not prevent expressions from evaluating to a result. Such a result is achieved by introducing a finite evaluation relation indexed over natural numbers. A terminating expression is one for which there exists a natural number n such that the finite evaluation indexed by n returns a value (which may include also the usual runtime errors). However, in our approach type soundness can be proved without introducing extra rules for dealing with runtime errors generation and propagation, and finite evaluations.

References

1. M. S. Ager. From natural semantics to abstract machines. In *LOPSTR*, pages 245–261, 2004.
2. R. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993.
3. D. Ancona. Coinductive big-step operational semantics for type soundness of Java-like languages. In *Formal Techniques for Java-like Programs*, FTfJP '11, pages 5:1–5:6. ACM, 2011.
4. D. Ancona, A. Corradi, G. Lagorio, and F. Damiani. Abstract compilation of object-oriented languages into coinductive CLP(X): can type inference meet verification? In B. Beckert and C. Marché, editors, *Post-proceedings of Formal Verification of Object-Oriented Software (FoVeOOS 2010)*, volume 6528 of *Lecture Notes in Computer Science*. Springer Verlag, 2011. Selected paper.
5. D. Ancona and G. Lagorio. Coinductive type systems for object-oriented languages. In S. Drossopoulou, editor, *ECOOP'09 - Object-Oriented Programming*, volume 5653 of *Lecture Notes in Computer Science*, pages 2–26. Springer Verlag, 2009. Best paper prize.
6. D. Ancona and G. Lagorio. Coinductive subtyping for abstract compilation of object-oriented languages into Horn formulas. In Montanari A., Napoli M., and Parente M., editors, *Proceedings of GandALF 2010*, volume 25 of *Electronic Proceedings in Theoretical Computer Science*, pages 214–223, 2010.
7. D. Ancona and G. Lagorio. Idealized coinductive type systems for imperative object-oriented programs. *RAIRO - Theoretical Informatics and Applications*, 45(1):3–33, 2011.
8. A. Arnold and M. Nivat. The metric space of infinite trees. Algebraic and topological properties. *Fundamenta Informaticae*, 3:445–476, 1980.
9. B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.
10. E. Ernst, K. Ostermann, and W.R. Cook. A virtual class calculus. In *POPL*, pages 270–282, 2006.
11. A. Igarashi, B. C. Pierce, and P. Wadler. Featherweight Java: a minimal core calculus for Java and GJ. *ACM Transactions on Programming Languages and Systems*, 23(3):396–450, 2001.
12. J. D. M. Kusmierek and V. Bono. Big-step operational semantics revisited. *Fundam. Inform.*, 103(1-4):137–172, 2010.
13. X. Leroy and H. Grall. Coinductive big-step operational semantics. *Information and Computation*, 207:284–304, 2009.
14. Tofte M. Milner R. Co-induction in relational semantics. *Theoretical Computer Science*, 87(1):209–220, 1990.
15. K. Nakata and T. Uustalu. Trace-based coinductive operational semantics for while. In *TPHOLs 2009*, pages 375–390, 2009.
16. K. Nakata and T. Uustalu. A Hoare logic for the coinductive trace-based big-step semantics of while. In *ESOP 2010*, pages 488–506, 2010.
17. L. Simon, A. Mallya, A. Bansal, and G. Gupta. Coinductive logic programming. In *Logic Programming, 22nd International Conference, ICLP 2006*, pages 330–345, 2006.
18. A. Stoughton. An operational semantics framework supporting the incremental construction of derivation trees. *Electr. Notes Theor. Comput. Sci.*, 10, 1997.