Modeling Infinite Behaviour by Corules

Davide Ancona[1]
DIBRIS, University of Genova, Italy
davide.ancona@unige.it
https://orcid.org/0000-0002-6297-2011

Francesco Dagnino
DIBRIS, University of Genova, Italy
francesco.dagnino@dibris.unige.it
https://orcid.org/0000-0003-3599-3535

Elena Zucca
DIBRIS, University of Genova, Italy
elena.zucca@unige.it
https://orcid.org/0000-0002-6833-6470

Abstract

Generalized inference systems have been recently introduced, and used, among other applications, to define semantic judgments which uniformly model terminating computations and divergence. We show that the approach can be successfully extended to more sophisticated notions of infinite behaviour, that is, to express that a diverging computation produces some possibly infinite result. This also provides a motivation to smoothly extend the theory of generalized inference systems to include, besides *coaxioms*, also *corules*, a more general notion for which significant examples were missing until now. We first illustrate the approach on a λ-calculus with output effects, for which we also provide an alternative semantics based on standard notions, and a complete proof of the equivalence of the two semantics. Then, we consider a more involved example, that is, an imperative Java-like language with I/O primitives.

2012 ACM Subject Classification Theory of computation → Operational semantics

Keywords and phrases Operational semantics, coinduction, trace semantics

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2018.21

1 Introduction

In semantic definitions of programming languages or systems, *finite* behaviour can be easily modeled by inductive techniques. For instance, both in small-step and big-step operational semantics, the fact that evaluation of an expression e terminates producing a final result r can be formally expressed as a judgment $e \Rightarrow r$ defined by an inference system[2], so that each valid judgment has a finite proof tree.

However, modeling *infinite* behaviour is not so easy. The simplest infinite behaviour we may want to model is divergence in itself, that is, the fact that evaluation of e does not terminate, as can be formally expressed by introducing a special result ∞. Traditionally, this is modeled at the meta-level in small-step definitions: the inference system does *not* define the judgment $e \Rightarrow \infty$, but only single steps, and then we formalize divergence simply as "an

[1] Member of GNCS (Gruppo Nazionale per il Calcolo Scientifico), INdAM (Istituto Nazionale di Alta Matematica "F. Severi")
[2] In small-step style this judgment can be inductively defined on top of the one-step reduction relation.

infinite sequence of steps". In big-step semantics the situation is even worse: non terminating and stuck computations are identified, since in both cases it is not possible to construct a finite proof tree showing that the computation returns a result.

A simple solution, in both approaches, is to define $e \Rightarrow \infty$ by a separate inference system, where inference rules are interpreted coinductively [8, 13]. However, in this way there are two stratified systems with overlapping rules, and there is no unique formal definition of the behaviour. The possibility of providing such unique definition has been investigated in previous work [13, 3], by interpreting coinductively the standard rules for the big-step operational semantics (*coevaluation*). Unfortunately, this approach is not satisfactory; for instance, coevaluation is non-deterministic in some cases: for a diverging term such as $\Omega = (\lambda x.x\ x)(\lambda x.x\ x)$, $\Omega \Rightarrow r$ can be derived for any r, instead of the unique valid judgment $\Omega \Rightarrow \infty$. Moreover, for some other diverging term, e.g., $\Omega\ (0\ 0)$, no judgment can be derived, since there is no valid judgment for the subterm $0\ 0$. This happens because in an application $e_1\ e_2$, if we follow a left-to-right strategy, then divergence of e_1 should be *propagated* regardless of e_2, but this cannot be properly modeled due to the presence of spurious judgments $\Omega \Rightarrow r$. In some recent work [6], it has been shown that both problems can be solved by defining the judgment $e \Rightarrow r$ by a *generalized inference system* [5]. This semantics can be seen as a *filtered coevaluation*. Indeed, infinite proof trees are allowed, but appropriate *coaxioms* are introduced to filter out spurious judgments $\Omega \Rightarrow r$, so that, for all terms expected to diverge, it is only possible to derive ∞ as result. In this way, it is also possible to provide rules for propagation of divergence, solving the second problem.

In the present paper, we face the problem of expressing more sophisticated notions of infinite behaviour. That is, we want to model not only that evaluation of an expression e could not terminate, but also that this diverging computation could produce some possibly infinite result. For instance, if computations can have output effects because of expression `out` e, then the result of a diverging computation could be a possibly infinite stream of output values. More in general, the behaviour of a non terminating program is a possibly infinite trace of observable events. As discussed above for pure divergence, this is modeled at the meta-level in small-step definitions: the inference system does *not* define the judgment $e \Rightarrow r$ where r is an infinite result, but only single labelled steps, and then we define divergent results as infinite sequences of labelled steps. On the other hand, again analogously to the pure divergence case, big-step rules interpreted simply coinductively allow derivation of spurious results. Hence, we investigate whether the approach of [6] could be effectively used also in this more general case to filter out such judgments.

Our conclusion, illustrated in the body of the paper, is that the approach based on filtered coevaluation works very well indeed for modeling trace semantics, or, more in general, infinite behaviour of divergent programs. Moreover, an interesting result is that, to filter out spurious judgments, we need to use not only coaxioms, but also *corules*. Corules were not considered in the original definition of generalized inference systems [5], even though formal definitions trivially extend to include them, simply for the lack of significant examples. Modeling infinite behaviour offers exactly such a significant example: notably, corules are needed to ensure that a diverging computation yields a possibly infinite result. For instance, if expression `out` e is reached infinitely many times during the non terminating evaluation of another expression, then the latter actually produces the effects of `out` e, providing that e converges. Similar corules are needed for other constructs having other kinds of observable behavior. Motivated by this important application, we present in this paper a slight variation of generalized inference systems [5] which also includes corules; we will use "generalized inference system" and "inference system with corules" as synonyms.

Summarizing the above discussion, the novel contribution of this paper w.r.t. previous work is not the general framework, which is the same of [5, 6] (except for the immediate generalization from coaxioms to corules), including, e.g., the fixpoint theory, not reported here. The novel, non trivial problem faced here is how to obtain, starting from an inductive judgment defining the result of finite computations, an extended one defining the (possibly infinite) result of infinite computations as well. We show that the problem can be solved by adding suitable corules and considering the resulting generalized inference system. This has been done in [6] only in the very special case where the only observable result is divergence. The general case is more challenging and requires more complex corules.

The paper is structured as follows. Sect. 2 introduces inference systems with corules, briefly recalling/extending the notions and results from [5]. In Sect. 3 we illustrate the approach on a lambda calculus enriched with output effects. That is, we show that, by using corules, we can directly define a unique judgment $e \Rightarrow r$ where r is either a finite result (final value and finite output stream) or an infinite result (∞ and possibly infinite output stream). In Sect. 4 we show that, by only using standard techniques, namely, labeled transition systems, coinduction, and observational equivalence, we can provide an alternative semantics which, however, requires much more work. In Sect. 5 we formally prove the equivalence of such two semantics. In Sect. 6 we consider a more involved example, that is, a simple imperative Java-like language with I/O primitives. Finally, the most relevant related work is surveyed in Sect. 7, and Sect. 8 concludes and outlines directions for further investigation. The Appendix contains an algebraic presentation of the observational equivalence, some of the proofs and additional examples.

2 Inference systems with corules

In this section, we provide a short introduction to inference systems with corules, needed to make the paper self-contained. The material is largely taken from [5], apart that we consider, besides coaxioms, *corules* with an analogous meaning. Here we focus on the proof-theoretic view, which is essential for our aims.

First of all we recall standard notions about inference systems [1, 13]. Assume a set $\mathcal{U}$ called the *universe*, whose elements are called *judgments*. An *inference system* $\mathcal{I}$ consists of a set of *(inference) rules*, which are pairs $\dfrac{Pr}{c}$, with $Pr \subseteq \mathcal{U}$ the set of *premises*, $c \in \mathcal{U}$ the *consequence* (a.k.a. *conclusion*). A rule with an empty set of premises is also called an *axiom*. A *proof tree* is a tree whose nodes are (labeled with) judgments in $\mathcal{U}$, and there is a node c with set of children Pr only if there exists a rule $\dfrac{Pr}{c}$.

The *inductive interpretation* of $\mathcal{I}$, denoted $[\![\mathcal{I}]\!]^{\mathsf{ind}}$, is the set of judgments which are the root of a finite[3] proof tree, whereas the *coinductive interpretation* of $\mathcal{I}$, denoted $[\![\mathcal{I}]\!]^{\mathsf{coind}}$, is the set of judgments which are the root of an arbitrary (finite or infinite) proof tree.

Both interpretations can also be characterized set-theoretically as follows. We define the *(one step) inference* operator $F_{\mathcal{I}} \colon \wp(\mathcal{U}) \to \wp(\mathcal{U})$ by $F_{\mathcal{I}}(S) = \{c \mid Pr \subseteq S, \dfrac{Pr}{c} \in \mathcal{I}\}$. A set S is *closed* if $F_{\mathcal{I}}(S) \subseteq S$, and *consistent* if $S \subseteq F_{\mathcal{I}}(S)$. That is, no new judgments can be inferred from a closed set, and all judgments in a consistent set can be inferred from the set itself. Then, it can be proved that $[\![\mathcal{I}]\!]^{\mathsf{ind}}$ is the smallest closed set, that is, the intersection of all closed sets, and $[\![\mathcal{I}]\!]^{\mathsf{coind}}$ is the largest consistent set, that is, the union of all consistent sets.

[3] Under the common assumption that sets of premises are finite, otherwise we should say a well-founded tree, that is, a tree with no infinite paths.

We describe now the notion of inference system with corules.

▶ **Definition 1** (Inference system with corules)**.** An *inference system with corules*, or *generalized inference system*, is a pair $(\mathcal{I}, \mathcal{I}^{\mathsf{co}})$ where $\mathcal{I}$ and $\mathcal{I}^{\mathsf{co}}$ are inference systems, whose elements are called *rules* and *corules*, respectively.

Analogously to rules, the meaning of corules is to derive a consequence from the premises. However, they can only be used in a special way, described in the following.

Given two inference systems $\mathcal{I}$ and $\mathcal{I}^{\mathsf{co}}$, $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$ is the (standard) inference system whose rules are the union of those in $\mathcal{I}$ and $\mathcal{I}^{\mathsf{co}}$. Moreover, given a subset S of the universe, $\mathcal{I}_{|S}$ denotes the inference system obtained from $\mathcal{I}$ by keeping only rules with consequence in S. Then, the interpretation of an inference system with corules $(\mathcal{I}, \mathcal{I}^{\mathsf{co}})$ is defined as follows.

1. First, we consider the inference system $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$ where corules can be used as rules as well, and we take its inductive interpretation $[\![\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}]\!]^{\mathsf{ind}}$.
2. Then, we take the coinductive interpretation of the inference system obtained from $\mathcal{I}$ by keeping only rules with consequence in $[\![\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}]\!]^{\mathsf{ind}}$.

Altogether, we get the following definition.

▶ **Definition 2** (Interpretation of a generalized inference system)**.** Let $(\mathcal{I}, \mathcal{I}^{\mathsf{co}})$ be a generalized inference system. Then, its *interpretation* $[\![\mathcal{I}, \mathcal{I}^{\mathsf{co}}]\!]$ is defined by

$$[\![\mathcal{I}, \mathcal{I}^{\mathsf{co}}]\!] = [\![\mathcal{I}_{|[\![\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}]\!]^{\mathsf{ind}}}]\!]^{\mathsf{coind}}$$

In proof-theoretic terms, $[\![\mathcal{I}, \mathcal{I}^{\mathsf{co}}]\!]$ is the set of judgments which have an arbitrary (finite or infinite) proof tree in $\mathcal{I}$, whose nodes all have a finite proof tree in $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$. Note that a finite proof tree in $\mathcal{I}$ is a finite proof tree in $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$ as well, hence the condition is only significant for nodes which are roots of an infinite path in the proof tree.

We report now some examples from [5] which illustrate the expressive power of generalized inference systems. Both examples only use corules with no premises (*coaxioms*). Examples which need corules with premises will be shown in the following section.
As usual, sets of rules can be expressed by a *metarule* with side conditions, and analogously sets of corules can be expressed by a *metacorule* with side conditions. In the examples, (meta)corules will be written $\dfrac{Pr}{\mathbf{c}}$, that is, with thicker lines, to be distinguished from (meta)rules.

The first example computes the judgment $n \xrightarrow{\star} \mathcal{N}$ meaning that $\mathcal{N}$ is the set of nodes reachable from a node n of a given graph. Let us represent a graph by its set of nodes V and a function adj which returns all the adjacents of a node. The judgment is defined by the generalized inference system $(\mathcal{I}, \mathcal{I}^{\mathsf{co}})$ where $\mathcal{I}$ and $\mathcal{I}^{\mathsf{co}}$ are all the instances of (ADJ) and (CO-EMPTY) below, respectively.

$$(\text{ADJ})\ \frac{n_1 \xrightarrow{\star} \mathcal{N}_1 \ \ \ldots \ \ n_k \xrightarrow{\star} \mathcal{N}_k}{n \xrightarrow{\star} \{n\} \cup \mathcal{N}_1 \cup \ldots \cup \mathcal{N}_k}\ adj(n) = \{n_1, \ldots, n_k\} \qquad (\text{CO-EMPTY})\ \frac{}{\mathbf{n \xrightarrow{\star} \emptyset}}\ n \in V$$

Consider, for instance, a graph with nodes a, b, c, with an arc from a into b and conversely, and c isolated. To show the aim of corules, let us first consider what happens if we only consider metarule (ADJ), disregarding the corules. We have in this way a (standard) inference system, which, as described above, can be interpreted either inductively or coinductively. However, neither interpretation provides the desired meaning. Indeed, if we interpret (ADJ) inductively, then we get only the judgment $c \xrightarrow{\star} \{c\}$. On the other hand, if we interpret the rules coinductively, then we get the correct judgments $a \xrightarrow{\star} \{a, b\}$ and $b \xrightarrow{\star} \{a, b\}$, but we also

get the wrong judgments $a \xrightarrow{\star} \{a, b, c\}$ and $b \xrightarrow{\star} \{a, b, c\}$. For instance, the judgment $a \xrightarrow{\star} \{a, b, c\}$ has the infinite proof tree shown below.

$$\dfrac{\dfrac{\dfrac{\vdots}{a \xrightarrow{\star} \{a, b, c\}}\,(\text{ADJ})}{b \xrightarrow{\star} \{a, b, c\}}\,(\text{ADJ})}{a \xrightarrow{\star} \{a, b, c\}}\,(\text{ADJ})$$

If we take into account corules, instead, then the interpretation of the resulting generalized inference system provides the desired meaning. For instance, in the example, the judgment $a \xrightarrow{\star} \{a, b\}$ has an infinite proof tree in $\mathcal{I}$ where each node has a finite proof tree in $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$, as shown below:

$$\dfrac{\dfrac{\dfrac{\vdots}{a \xrightarrow{\star} \{a, b\}}\,(\text{ADJ})}{b \xrightarrow{\star} \{a, b\}}\,(\text{ADJ})}{a \xrightarrow{\star} \{a, b\}}\,(\text{ADJ}) \qquad \dfrac{\dfrac{\dfrac{}{a \xrightarrow{\star} \emptyset}\,(\text{CO-EMPTY})}{b \xrightarrow{\star} \{b\}}\,(\text{ADJ})}{a \xrightarrow{\star} \{a, b\}}\,(\text{ADJ}) \qquad \dfrac{\dfrac{\dfrac{}{b \xrightarrow{\star} \emptyset}\,(\text{CO-EMPTY})}{a \xrightarrow{\star} \{a\}}\,(\text{ADJ})}{b \xrightarrow{\star} \{a, b\}}\,(\text{ADJ})$$

whereas this is not the case for the judgment $a \xrightarrow{\star} \{a, b, c\}$. In other words, corules filter out undesired infinite proof trees.

Note that the inductive and coinductive interpretation of $\mathcal{I}$ are special cases, notably:

- the inductive interpretation of $\mathcal{I}$ is the interpretation of $(\mathcal{I}, \emptyset)$
- the coinductive interpretation of $\mathcal{I}$ is the interpretation of $(\mathcal{I}, \{\dfrac{\emptyset}{c} \mid c \in \mathcal{U}\})$.

In [5] it is shown that this corresponds to taking a fixed point of $F_{\mathcal{I}}$ which is, in general, neither the least, nor the greatest.

We show now how the recursive definition of a function can be expressed as an inference system with corules. Let $\mathbb{Z}$ denote the set of integers, $\mathbb{L}$ the set of (finite and infinite) lists of integers, Λ the empty list and $x{:}l$ the list with head x and tail l.

The function which returns the greatest element contained in a (non empty) list is expressed by judgments of shape $max(l, x)$, with $l \in \mathbb{L}$ and $x \in \mathbb{Z}$.

$$\dfrac{}{max(x{:}\Lambda, x)} \qquad \dfrac{max(l, y)}{max(x{:}l, z)}\, z = \max(x, y) \qquad \dfrac{}{max(x{:}l, x)}$$

Without coaxioms the coinductive interpretation fails to be a function (for instance, for l the infinite list of 1s, any judgment $max(l, x)$ with $x \geq 1$ can be derived), and the coaxioms "filter out" the wrong results. We refer to related work [5, 6] for other examples.

Several proof techniques have been proposed for generalized inference systems with coaxioms [5]. For the aim of this paper, we only need the *bounded coinduction principle* reported below, which is a generalization of the standard coinduction principle.

Let $(\mathcal{I}, \mathcal{I}^{\mathsf{co}})$ be a generalized inference system, and $\mathcal{S}$ (for "specification") an intended set of judgments, called *valid* in the following. *Completeness*, that is, the property that each valid judgment can be derived ($\mathcal{S} \subseteq [\![\mathcal{I}, \mathcal{I}^{\mathsf{co}}]\!]$), can be proved as follows:

▶ **Theorem 3** (Bounded coinduction principle). *If the following two conditions hold:*

1. $\mathcal{S} \subseteq [\![\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}]\!]^{ind}$*, that is, each valid judgment has a finite proof tree in* $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$*;*
2. $\mathcal{S} \subseteq F_{\mathcal{I}}(\mathcal{S})$*, that is, each valid judgment is the consequence of an inference rule in* $\mathcal{I}$ *where all premises are in* $\mathcal{S}$

then $\mathcal{S} \subseteq [\![\mathcal{I}, \mathcal{I}^{\mathsf{co}}]\!]$.

$$
\begin{array}{rcll}
e & ::= & v \mid x \mid e_1\ e_2 \mid \mathtt{out}\ e & \text{expression} \\
v & ::= & i \mid \lambda x.e & \text{value} \\
\ell & ::= & v \mid \tau & \text{label} \\
\mathcal{E}[\,] & ::= & \Box \mid \mathcal{E}[\,]\ e \mid v\ \mathcal{E}[\,] \mid \mathtt{out}\ \mathcal{E}[\,] & \text{evaluation context}
\end{array}
$$

$$
(\beta)\frac{}{(\lambda x.e)\ v \xrightarrow{\tau} e[x \leftarrow v]} \qquad (\text{OUT})\frac{}{\mathtt{out}\ v \xrightarrow{v} v} \qquad (\text{CTX})\frac{e \xrightarrow{\ell} e'}{\mathcal{E}[e] \xrightarrow{\ell} \mathcal{E}[e']}\ \mathcal{E}[\,] \neq \Box
$$

Figure 1 λ-calculus with output effects: syntax and labeled transition system.

The standard coinduction principle can be obtained when $\mathcal{I}^{\mathsf{co}} = \{\frac{\emptyset}{c} \mid c \in \mathcal{U}\}$; for this particular case the first condition trivially holds.

3 Infinite behaviour by corules

As mentioned in the Introduction, in this paper we are interested in modeling infinite behaviour. That is, in addition to explicitly model divergent computations, we would like to also model their possibly infinite result. This infinite result can be thought of as what an external observer can see during the infinite computation. In this section we show how corules can be used to define such a semantics.

We illustrate the technique by an extension of the call-by-value lambda calculus with output effects. The syntax is given in Fig. 1, together with a labeled transition system meant to provide the reader with a simple formal account of the intended meaning.

We assume infinite sets of *variables* x and *integer constants* i. Values are defined in the standard way, as either integer literals or lambda abstractions. Beyond standard constructs, we added expressions of shape $\mathtt{out}\ e$, which output the result of the evaluation of e.

As it is common practice, the reduction relation is decorated by a *label* ℓ representing the observable effect of a single step. A label v represents an output effect, and can be generated by rule (OUT), while a label τ represents the absence of observable output and can be generated by the β-rule. The rule (CTX) is the usual contextual closure and defines the standard (call-by-value and left-to-right) evaluation strategy.

As discussed in the Introduction, the labelled transition system in Fig. 1 models a single evaluation step, and infinite behaviour is only obtained at the meta-level by considering infinite sequences of labelled steps.

We show now a generalized inference system defining a judgment $e \Rightarrow r$ which directly models both finite and infinite behaviour of expressions.

The top section of Fig. 2 defines *results* r of (finite and infinite) computations. If the evaluation of an expression terminates, then the result of the computation is of shape (v, o) where v is the final value, and o is the (necessarily finite) output stream produced during the evaluation. If the evaluation of an expression diverges, hence there is no final value, then the result is of shape (∞, o_∞) where o_∞ is the (possibly infinite) output stream produced during the evaluation. Output streams are sequences of values delimited by square brackets for readability. Streams grow left-to-right; hence, the rightmost element in a finite stream corresponds to the most recent output value. Concatenation of two streams $o \cdot o_\infty$ is defined in the usual way, under the assumption that the left-hand side operand must be finite.

The second section of Fig. 2 contains the generalized inference system defining the judgment $e \Rightarrow r$, meaning that r is the (finite or infinite) result of the evaluation of e.

$$
\begin{array}{rcll}
r & ::= & (v, o) \mid (\infty, o_\infty) & \text{result} \\
o & ::= & [v_1 \dots v_n] & \text{finite output} \\
o_\infty & ::= & o \mid [v_1 \dots v_n \dots] & \text{finite/infinite output} \\
\mathcal{D}[\,] & ::= & \Box\ e \mid \Box\ \Box \mid \mathtt{out}\ \Box & \text{(divergence) propagation context}
\end{array}
$$

$$
(\textsc{val})\ \frac{}{v \Rightarrow (v, [\,])} \qquad (\textsc{app})\ \frac{e_1 \Rightarrow (\lambda x.e, o_1) \quad e_2 \Rightarrow (v, o_2) \quad e[x \leftarrow v] \Rightarrow r}{e_1\ e_2 \Rightarrow o_1 \cdot o_2 \cdot r}
$$

$$
(\textsc{out})\ \frac{e \Rightarrow (v, o)}{\mathtt{out}\ e \Rightarrow (v, o \cdot [v])} \qquad (\textsc{div})\ \frac{\forall\, i = 1..n. e_i \Rightarrow (v_i, o_i) \quad e \Rightarrow (\infty, o_\infty)}{\mathcal{D}[\overline{e}^n, e] \Rightarrow (\infty, o_1 \cdot \ldots \cdot o_n \cdot o_\infty)}
$$

$$
(\textsc{co-empty})\ \frac{}{e \Rightarrow (\infty, [\,])} \qquad (\textsc{co-out})\ \frac{e \Rightarrow (v, o)}{\mathtt{out}\ e \Rightarrow (\infty, o \cdot [v] \cdot o_\infty)}
$$

Figure 2 λ-calculus with output effects: inference system with corules.

Propagation contexts can have one or two holes (all at fixed depth 1) and allow a more concise treatment of divergence propagation with a single rule, see comments below for (DIV).

We recall that in rules thicker lines distinguish corules from rules.

Rule (VAL) is straightforward: the evaluation of a value always converges to itself and produces no output.

Rule (APP) is an extension of the usual big-step rule for application. First, the two subexpressions are evaluated; if they both converge, then the body of the function is evaluated after the formal parameter has been substituted with its argument. As usual, $e[x \leftarrow v]$ denotes capture-avoiding substitution modulo α-renaming. The evaluation of the body returns a general result, meaning that the evaluation may either converge or diverge. The final result is obtained by suitably concatenating the output streams generated by the evaluations of e_1 and e_2 with that generated by the evaluation of the body. If $r = (v_\infty, o_\infty)$, then $o \cdot r$ denotes $(v_\infty, o \cdot o_\infty)$, where v_∞ is either a value v or ∞.

In rule (OUT), if the evaluation of e converges to a value v, then the whole expression converges to the same value; moreover, the value v is added to the output stream o generated by the evaluation of e.

The remaining rule (DIV) deals with propagation of non termination and composition of the corresponding output streams. Evaluation of the subexpressions in the holes of the context proceeds from left to right and the subexpression corresponding to the rightmost hole is the first one which diverges. In this simple language (divergence) propagation contexts have no more than two holes, therefore the number n of subexpressions that converge can be either 0 or 1. More in detail, the context $\Box\ e$ corresponds to the case where application diverges because the left-hand side expression does not terminate, whereas the context $\Box\ \Box$ to the case where the left-hand side expression terminates, whereas the right-hand side does not. Finally, $\mathtt{out}\ \Box$ propagates non termination of the unique subexpression of $\mathtt{out}\ e$.

As explained in the previous section, corules are used to filter out spurious results of divergent computations.

Corule (CO-EMPTY) deals with divergent computations which produce a finite output stream, hence do not produce any output (we will also say that the infinite computation is "non-productive") from a certain point. In this case, as shown in the first two examples below, (CO-EMPTY) prevents derivation of judgments with arbitrary output streams, similarly to the examples shown in the previous section. In all such cases the only correct option is non-termination with the empty output stream $[\,]$.

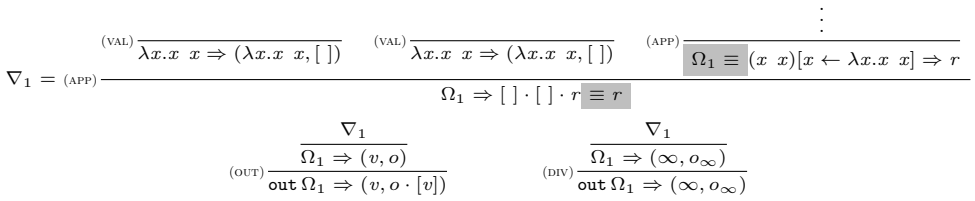


Figure 3 Infinite proof trees for Example 1.

Corule (CO-OUT) deals with non terminating terms which produce an infinite number of values; this happens if expressions of shape $\texttt{out}\ e$ are evaluated an infinite number of times. The premise requires the subexpression e to converge to a value v, ensuring that the output expression is actually evaluated, adding v to the output stream. In this way we guarantee productivity, that may not hold if e diverges (see the difference between examples 2 and 3 below).

Example 1. As a first example, we consider the term $\texttt{out}\,\Omega_1$, where $\Omega_1 = (\lambda x.x\ x)\ (\lambda x.x\ x)$; the only derivable judgment is $\texttt{out}\,\Omega_1 \Rightarrow (\infty, [\,])$, corresponding to the expected semantics: the evaluation of $\texttt{out}\,\Omega_1$ diverges and generates an empty output stream.

Indeed, a judgment $\texttt{out}\,\Omega_1 \Rightarrow r$ is derivable if:

- it has a possibly infinite proof tree with no corules
- such proof tree is valid according to the corules, that is, each node has a finite proof tree with corules.

The first condition is illustrated in Fig. 3. The top part of the figure shows the infinite proof tree for the judgment $\Omega_1 \Rightarrow r$, for all possible r, where the vertical dots indicate that the proof continues with the same repeated pattern. Here and in the following examples, we add to the proof tree some comments (with a grey background) showing an equivalent expression, as a help for the reader. No other finite or infinite proof trees can be built for such judgment. In such a simple case the proof tree is regular; however, there exist examples of divergent computations with non regular proof trees.

For the evaluation of $\texttt{out}\,\Omega_1$ we can apply two different rules, depending on the shape of r. If r is a converged result (v, o), then the only applicable rule is (OUT), and we derive the judgment $\texttt{out}\,\Omega_1 \Rightarrow (v, o \cdot [v])$; otherwise, r is a diverged result (∞, o_∞), and the judgment $\texttt{out}\,\Omega_1 \Rightarrow (\infty, o_\infty)$ can be derived by rule (DIV).

Among all judgments derivable with an infinite tree built with only the rules as shown above, the only one that is valid according to the corules is $\texttt{out}\,\Omega_1 \Rightarrow (\infty, [\,])$; in this case it suffices to exhibit a finite proof tree for $\Omega_1 \Rightarrow (\infty, [\,])$ which can be built by applying also the corules. Such a tree is trivial, thanks to coaxiom (CO-EMPTY):

$$\text{(CO-EMPTY)}\ \frac{}{\Omega_1 \Rightarrow (\infty, [\,])}$$

By rule (DIV), and from the trivial finite tree above, it is possible to derive a finite tree also for the judgment $\texttt{out}\,\Omega_1 \Rightarrow (\infty, [\,])$.

It is easy to see that instead, for $r \neq (\infty, [\,])$, it is not possible to derive a finite tree, built by also the corules, for the judgment $\Omega_1 \Rightarrow r$.

$$\nabla_2 = \dfrac{\dfrac{}{\omega_2 \Rightarrow (\omega_2, [\,])}\,(\text{VAL}) \quad \dfrac{}{\omega_2 \Rightarrow (\omega_2, [\,])}\,(\text{VAL}) \quad \dfrac{\dfrac{\vdots}{\Omega_2 \Rightarrow (\infty, o_\infty)}\,(\text{APP})}{\mathtt{out}\,\Omega_2 \equiv (\mathtt{out}\ (x\ \ x))[x \leftarrow \omega_2] \Rightarrow (\infty, o_\infty)}\,(\text{DIV})}{\Omega_2 \Rightarrow [\,] \cdot [\,] \cdot (\infty, o_\infty) \equiv (\infty, o_\infty)}\,(\text{APP})$$

Figure 4 Infinite proof trees for Example 2.

$$\nabla_3 = \dfrac{\dfrac{}{\omega_3 \Rightarrow (\omega_3, [\,])}\,(\text{VAL}) \quad \dfrac{\dfrac{}{\omega_3 \Rightarrow (\omega_3, [\,])}\,(\text{VAL})}{\mathtt{out}\ \omega_3 \Rightarrow (\omega_3, [\omega_3])}\,(\text{OUT}) \quad \dfrac{\vdots}{\omega_3\ \ (\mathtt{out}\ \omega_3) \equiv (x\ \ (\mathtt{out}\ x))[x \leftarrow \omega_3] \Rightarrow (\infty, o_\infty)}\,(\text{APP})}{\omega_3\ \ (\mathtt{out}\ \omega_3) \equiv (x\ \ (\mathtt{out}\ x))[x \leftarrow \omega_3] \Rightarrow [\,] \cdot [\omega_3] \cdot (\infty, o_\infty)}\,(\text{APP})$$

$$\dfrac{\dfrac{}{\omega_3 \Rightarrow (\omega_3, [\,])}\,(\text{VAL}) \quad \dfrac{}{\omega_3 \Rightarrow (\omega_3, [\,])}\,(\text{VAL}) \quad \nabla_3}{\Omega_3 \Rightarrow [\,] \cdot [\,] \cdot (\infty, o_\infty) \equiv (\infty, o_\infty)}\,(\text{APP})$$

Figure 5 Infinite proof tree for Example 3.

Example 2. As a second example, we consider the term $\Omega_2 = \omega_2\ \omega_2 = (\lambda x.\mathtt{out}\,(x\ \ x))\ (\lambda x.\mathtt{out}\,(x\ \ x))$ and show that, as in the previous case, the only derivable judgment is $\Omega_2 \Rightarrow (\infty, [\,])$, as expected: the evaluation of Ω_2 does not terminate and does not output any value.

By applying only the rules, the infinite proof tree shown in Fig. 4 can be built for the judgment $\Omega_2 \Rightarrow (\infty, o_\infty)$, for any possible o_∞. No judgments of shape $\Omega_2 \Rightarrow (v, o)$ can be derived, because this could be achieved only with an infinite proof tree containing infinite applications of rule (OUT) for the judgment $\mathtt{out}\,\Omega_2 \Rightarrow (v, o)$. This is not possible because such a rule is applicable only for finite output streams, whereas the infinite applications of (OUT) would force o to be infinite. Among all judgments derivable with an infinite tree built with the rules as shown above, the only one that is valid according to the corules is $\mathtt{out}\,\Omega_2 \Rightarrow (\infty, [\,])$; in this case it is sufficient to exhibit a finite proof tree which uses also the corules for the judgment $\mathtt{out}\,\Omega_2 \Rightarrow (\infty, [\,])$. Again, this can be simply obtained by corule (CO-EMPTY):

$$\dfrac{}{\mathtt{out}\,\Omega_2 \Rightarrow (\infty, [\,])}\,(\text{CO-EMPTY})$$

By rule (APP), and from the simple finite tree above, it is possible to derive a finite tree for the judgment $\Omega_2 \Rightarrow (\infty, [\,])$ as well.

No finite proof tree can be derived for $\mathtt{out}\,\Omega_2 \Rightarrow (\infty, o_\infty)$ for any $o_\infty \neq [\,]$, because corule (CO-OUT) can be used only if $\Omega_2 \Rightarrow (v, o)$ can be derived (by using also the corules) with a finite tree, but this is not possible, because the only possible derivable trees are infinite, as shown above.

Example 3. As a final example, we show that the only judgment derivable for $\Omega_3 = \omega_3\ \ \omega_3 = (\lambda x.(x\ \ (\mathtt{out}\ x)))\ \ (\lambda x.(x\ \ (\mathtt{out}\ x)))$ is $\Omega_3 \Rightarrow (\infty, [\omega_3\ \ldots\ \omega_3\ \ldots])$, corresponding to the expected semantics: the evaluation of Ω_3 diverges and generates the output stream consisting of infinite occurrences of the value ω_3.

In the system without corules it is possible to build only one proof tree, shown in Fig. 5, which is infinite and forces the equation $o_\infty = [\omega_3] \cdot o_\infty$, which admits a unique solution; hence, the judgment $\Omega_3 \Rightarrow (\infty, o_\infty)$ is derivable only for $o_\infty = [\omega_3 \ \ldots \ \omega_3 \ \ldots]$. Then, we have to show that by considering also the corules we can derive finite proof trees for all judgments involved in the proof tree for $\Omega_3 \Rightarrow (\infty, o_\infty)$; to this aim, it is sufficient to exhibit a finite derivation just for the judgment $(x \ \ (\texttt{out}\ x))[x \leftarrow \omega_3] \Rightarrow (\infty, o_\infty)$ at the root of proof tree ∇_3.

$$\dfrac{\dfrac{}{\omega_3 \Rightarrow (\omega_3, [\,])}\,(\textsc{val}) \qquad \dfrac{\dfrac{}{\omega_3 \Rightarrow (\omega_3, [\,])}\,(\textsc{val})}{\texttt{out}\,(\omega_3) \Rightarrow (\infty, o_\infty)}\,(\textsc{co-out})}{(x \ \ (\texttt{out}\ x))[x \leftarrow \omega_3] \Rightarrow [\,] \cdot (\infty, o_\infty) \equiv (\infty, o_\infty)}\,(\textsc{div})$$

By rule (app) and from the finite tree above, it is possible to derive a finite tree as well for $\Omega_3 \Rightarrow (\infty, o_\infty)$.

We conclude this section by proving a *conservativity* property [6] which we always expect to hold for an operational semantics modeling also divergence, given through corules. Namely, we require that the introduction of divergence does not affect standard convergent computations, as formally stated in the following theorem. This property is important since it implies that, for convergent judgments, we can reason by standard inductive techniques.

▶ **Theorem 4** (Conservativity of $e \Rightarrow r$)**.** *If $e \Rightarrow r$ holds, then $r = (v, o)$ if and only if the judgment has a finite proof tree.*

Proof. Let us denote by $(\mathcal{I}, \mathcal{I}^{\mathsf{co}})$ the generalized inference system defining the judgment $e \Rightarrow r$ in Fig. 2, and by $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$ the (standard) inference system that is the union of $\mathcal{I}$ and $\mathcal{I}^{\mathsf{co}}$. If $e \Rightarrow r$ is derivable in $(\mathcal{I}, \mathcal{I}^{\mathsf{co}})$, then by definition all judgments in the proof tree have a finite derivation in $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$, including the judgment itself. Noting that a rule's conclusion is a divergent judgment iff at least one of the premises is also divergent (see rules (div) and (app)), it can be shown by induction on the depth of the tree that $r = (\infty, o_\infty)$ iff we use either (co-empty) or (co-out) in the tree. It follows that $r = (v, o)$ iff we do not use (co-empty) and (co-out) in the finite proof tree in $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$, hence it is a finite proof tree in $\mathcal{I}$ as needed. ◀

4 Infinite behaviour by standard techniques

In this section we show that, by only using standard techniques, namely, labeled transition systems (LTSs from now on), coinduction, and observational equivalence, we can provide an alternative semantics for lambda calculus with output effects, which, however, is much more involved than the direct definition provided in the previous section.

The starting point is the LTS introduced in Fig. 1. This (inductive) inference system provides a very simple and intuitive model, which, however, is not abstract enough for reasoning about the observable behaviour of programs. Namely, two expressions having different sequences of labeled steps could be equivalent in terms of their observable behaviour.

To obtain, starting from the LTS, the same level of abstraction of the semantics defined in the previous section, some additional work is needed.

First of all, as discussed in the Introduction, the overall result of a computation is only modeled at the meta-level in the labeled transition system: that is, this inference system only defines single steps, and then we say that there is "a possibly infinite sequence of labeled steps". To express this statement, instead, in a formal way, we can define, of top of the LTS, another judgment $e \rightsquigarrow \widetilde{r}$ which associates with each expression e its result, consisting of the final value, if any, or ∞ if diverging, and the possibly infinite stream of labels produced during the computation. This judgment can be defined by standard coinduction, as detailed below.

$$
\begin{array}{rcll}
\widetilde{r} & ::= & (v, \widetilde{o}) \mid (\infty, \widetilde{o}_\infty) & \text{(rough) result} \\
\widetilde{o}, \widetilde{s} & ::= & [\ell_1 \dots \ell_n] & \text{finite stream} \\
\widetilde{o}_\infty, \widetilde{s}_\infty & ::= & \widetilde{o} \mid [\ell_1 \dots \ell_n \dots] & \text{finite/infinite stream}
\end{array}
$$

$$
\frac{}{v \rightsquigarrow (v, [\,])} \qquad \frac{e' \rightsquigarrow \widetilde{r}}{e \rightsquigarrow [\ell] \cdot \widetilde{r}}\ e \xrightarrow{\ell} e' \qquad \frac{}{e \rightsquigarrow \widetilde{r}}
$$

Figure 6 λ-calculus with output effects: coinductive inference system.

However, not even this judgment is abstract enough, since, as mentioned above, different sequences of labels could be equivalent in terms of observable behaviour, that is, roughly, τ steps should be not relevant. In other words, results $\widetilde{r}$ obtained in this judgment need a further abstraction step, consisting in the definition of an appropriate observational equivalence: in the following, we will call them *rough* results, and we will call *observable* results those obtained in the judgment defined in the previous section. We describe now in details the two steps.

Computing rough results. The definition of the judgment $e \rightsquigarrow \widetilde{r}$ is given in Fig. 6.

The top section of the figure defines (rough) results $\widetilde{r}$ of (finite and infinite) computations. The definition is analogous to that of (observable) results r in Fig. 2. However, here the second component of a result is, rather than an output stream (a stream of values), a stream of labels (called simply a stream from now on), corresponding to the fact that a computation step can be silent (no output). Concatenation of two streams $\widetilde{o} \cdot \widetilde{o}_\infty$ is also defined analogously to that of output streams, hence requires that $\widetilde{o}$ is finite.

The second section of the figure contains the coinductive inference system defining the judgment $e \rightsquigarrow \widetilde{r}$, meaning that $\widetilde{r}$ is the (rough) finite or infinite result of the evaluation of e.

Again analogously to output streams and (observable) results, we can extend concatenation of streams to (rough) results by setting $\widetilde{o} \cdot (v_\infty, \widetilde{o}_\infty) = (v_\infty, \widetilde{o} \cdot \widetilde{o}_\infty)$, where v_∞ is either a value v or ∞.

A very important point to be noted and discussed is that the definition of the judgment $e \rightsquigarrow \widetilde{r}$ is *purely coinductive*. Indeed, in Fig. 6, we used the notation of generalized inference systems for uniformity, but, as mentioned on page 5, a generalized inference system with a unique (meta)coaxiom where the consequence is any judgment exactly corresponds to the inference system consisting of only the rules, interpreted coinductively.

In this case it is possible to give a purely coinductive definition, without incurring in the problem of spurious judgment discussed in the Introduction for coevaluation, since the definition is *productive*. That is, each time we apply the inference rule, we add a label ℓ to the previously computed result, thus also infinite derivations admit a unique result. In order to guarantee productivity, the presence of labels τ is essential, even though they are not interesting semantically, since they represent non-observable steps. This motivates the fact that, as mentioned above, by using standard techniques we need two additional steps on top of the labeled transition system: indeed, to be able to use pure coinduction we need rough results, and then we need to identify rough results which are observationally equivalent. By using corules, instead, as shown in the previous section, we are able to directly define the (observable) behaviour by a unique judgment.

$$(\text{TAU})\ \frac{}{\tau^\alpha \approx \tau^\beta}\ \alpha,\beta \in \mathbb{N}\cup\{\omega\} \qquad (\text{VAL})\ \frac{\widetilde{o}_\infty \approx \widetilde{o}'_\infty}{\tau^n\cdot[v]\cdot\widetilde{o}_\infty \approx \tau^m\cdot[v]\cdot\widetilde{o}'_\infty}\ n,m\in\mathbb{N} \qquad (\text{CO})\ \frac{}{\widetilde{o}_\infty \approx \widetilde{o}'_\infty}$$

Figure 7 Observational equivalence: coinductive inference system.

Also for the judgment $e \rightsquigarrow \widetilde{r}$ we can state a conservativity result analogous to Theorem 4, implying that, for convergent judgments, we can reason by standard inductive techniques. In this case, we have a one-to-one correspondence between finite proof trees and convergent computations, and between infinite proof trees and divergent computations.

▶ **Theorem 5** (Conservativity of $e \rightsquigarrow \widetilde{r}$)**.** *If $e \rightsquigarrow \widetilde{r}$ holds, then $\widetilde{r} = (v, \widetilde{o})$ if and only if the judgment has a finite proof tree.*

Proof. A finite proof tree for $e \rightsquigarrow \widetilde{r}$ needs to start from the axiom $v \rightsquigarrow (v, [\,])$, hence $\widetilde{r} = (v, \widetilde{o})$. On the other hand, an infinite proof tree for $e \rightsquigarrow \widetilde{r}$, since it adds a label to the result at each level, requires the output stream in the result to be infinite, therefore, thanks to the way we defined results, we get $\widetilde{r} = (\infty, \widetilde{o}_\infty)$. ◀

Observational equivalence. We discuss now how to define observational equivalence on streams. Intuitively, since τ labels represent non-observable actions, they should be ignored when comparing two streams: for instance, streams $[v_1\,\tau\,v_2]$ and $[v_1\,v_2]$ should be equivalent. Therefore we need to relax equality to another equivalence relation, denoted by $\approx$, coinductively defined by the inference system shown in Fig. 7, where τ^n is the stream of length n made of only τs and τ^ω is the infinite stream of τs.

The intuition behind this definition is that sequences of τ of arbitrary length are equivalent to $[\,]$ (as stated in rule (TAU)), hence they can be removed or added without changing the observable view (non-τ elements) of the stream. Note that the two rules are disjoint, since the consequence of the second rule requires an element of the stream to be different from τ on both sides, hence it may not happen that a stream made of only τs is equivalent to a stream that contains non-τ elements. As in Fig. 6, the unique (meta)coaxiom where the consequence is any judgment corresponds to take the coinductive interpretation of the two rules.

This relation is indeed an equivalence, as formally stated below (the proof is in the Appendix).

▶ **Proposition 6.** *The relation $\approx$ is an equivalence relation over (finite or infinite) streams.*

A more abstract view of the relation $\approx$ can be given in categorical terms, namely as a bisimulation on a coalgebra structure carried by the set of streams. For the interested reader, this alternative definition is reported, and shown to be equivalent to the previous one, in the Appendix, Sect. A. To follow the next section, it is important to know that, following this definition, two streams are identified if and only if they are mapped to the same output stream by a function ε_τ that removes τs from a stream, which can be described by the following equations:

$$\begin{cases} \varepsilon_\tau(\tau^\alpha) & = [\,] \\ \varepsilon_\tau(\tau^n\cdot[v]\cdot x) & = [v]\cdot\varepsilon_\tau(x) \end{cases}$$

Function ε_τ is employed in the next section to prove Theorem 7.

Now we have to extend this relation to results. Intuitively, two results are equivalent if they represent either both convergence with the same value, or both divergence, and the observable output streams are equivalent. More formally, we set

$$(v_\infty, \widetilde{o}_\infty) \approx (v'_\infty, \widetilde{o}'_\infty) \iff v_\infty = v'_\infty \text{ and } \widetilde{o}_\infty \approx \widetilde{o}'_\infty$$

It is immediate to check that this relation is also an equivalence. Note also that this extension can be defined by an inference system (interpreted coinductively) obtained from the definition of $\approx$ on streams, by decorating each stream with the same (extended) value v_∞, hence to prove $\widetilde{r} \approx \widetilde{r}'$ we can reason by coinduction.

Having introduced all this machinery, we can define that two expressions e and e' are *observationally equivalent* if $e \rightsquigarrow \widetilde{r}$ implies $e' \rightsquigarrow \widetilde{r}'$, for some $\widetilde{r}' \approx \widetilde{r}$, and conversely.

Consider, for instance, the value $id = \lambda x.x$ and the expressions $e_1 = \texttt{out}\,(id\ id)$ and $e_2 = (\texttt{out}\ id)\ id$. It is easy to see that the judgments $e_1 \rightsquigarrow (id, [\tau\ id])$ and $e_2 \rightsquigarrow (id, [id\ \tau])$ hold, and $[\tau\ id] \approx [id\ \tau]$, hence e_1 and e_2 are observationally equivalent.

Note that, considering the judgment $e \Rightarrow r$ defined in the previous section by using corules, the definition of observational equivalence reduces to semantic equivalence, that is, expressions e and e' are equivalent iff $e \Rightarrow r$ implies $e' \Rightarrow r$, and conversely. In other words, the judgment $e \Rightarrow r$ directly models the observable behaviour of programs. For instance, again it is easy to see that $e_1 \Rightarrow (id, [id])$ and $e_2 \Rightarrow (id, [id])$, hence e_1 and e_2 are equivalent also with respect to this semantics. Actually, as we will see in the next section, the two semantics can be shown to be equivalent.

5 Equivalence between the two semantics

In this section we will provide a complete[4] proof of the equivalence between the semantics defined in Fig. 2 by using corules and that defined in Fig. 6 on top of the LTS in Fig. 1. We will briefly call them semantics by corules and LTS semantics, respectively. The main theorem is stated below.

▶ **Theorem 7** (Equivalence).
1. *If $e \Rightarrow r$, then there exists $\widetilde{r}$ such that $e \rightsquigarrow \widetilde{r}$ and $r \approx \widetilde{r}$.*
2. *If $e \rightsquigarrow \widetilde{r}$, then there exists r such that $e \Rightarrow r$ and $\widetilde{r} \approx r$.*

We will prove separately the two parts of the theorem. Before providing such proofs we need to consider some properties relating the two semantics.

▶ **Lemma 8** (Progress). *If $e \Rightarrow r$ and e is not a value, then there exists e' and ℓ such that $e \xrightarrow{\ell} e'$.*

Proof. Straightforward induction on the definition of e. ◀

▶ **Lemma 9** (Subject reduction). *If $e \Rightarrow r$ and $e \xrightarrow{\ell} e'$, then $e' \Rightarrow r'$ and $r \approx [\ell] \cdot r'$.*

Proof. Straightforward induction on the rules defining $e \xrightarrow{\ell} e'$. ◀

The following lemma states that, if an expression converges in the semantics by corules and produces a (rough) result in the LTS semantics, then the LTS semantics converges as well with the same value.

[4] Proofs of some auxiliary results are in the Appendix.

▶ **Lemma 10.** *If $e \Rightarrow (v, o)$ and $e \rightsquigarrow \widetilde{r}$, then $\widetilde{r} = (v, \widetilde{o})$.*

Proof. Thanks to Theorem 4, we can reason by induction on the rules defining $e \Rightarrow (v, o)$, hence the proof is routine. ◀

In the next theorem, starting from a judgment $e_0 \Rightarrow r_0$ in the semantics by corules, we construct a sequence of reduction steps in the LTS, which can be either finite or infinite:

$$e_0 \xrightarrow{\ell_0} e_1 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_{n-1}} e_n \equiv v$$
$$e_0 \xrightarrow{\ell_0} e_1 \xrightarrow{\ell_1} \dots e_n \xrightarrow{\ell_n} e_{n+1} \xrightarrow{\ell_{n+1}} \dots$$

(for uniformity, a finite reduction sequence is represented by a reduction sequence until $e_n \equiv v$ and then an infinite sequence of v). For each n, we also construct a result r_n and a rough result $\widetilde{r}_n$ such that $e_n \Rightarrow r_n$, $e_n \rightsquigarrow \widetilde{r}_n$, and the first component (value or divergence) in r_n and $\widetilde{r}_n$ is the same. Thus, in particular, there exists $\widetilde{r}_0$ so that $e_0 \rightsquigarrow \widetilde{r}_0$ and the first component (value or divergence) in r_0 and $\widetilde{r}_0$ is the same.

▶ **Theorem 11.** *If $e_0 \Rightarrow r_0$, then there exist sequences $(e_n)_{n\in\mathbb{N}}$ of expressions, $(r_n)_{n\in\mathbb{N}}$ of results, and $(\widetilde{r}_n)_{n\in\mathbb{N}}$ of rough results, such that, for all $n \in \mathbb{N}$*

1. *$e_n \Rightarrow r_n$ and*
   - *if $e_n = v$, then $e_{n+1} = v$ and $r_n = r_{n+1} = \widetilde{r}_n = \widetilde{r}_{n+1} = (v, [\,])$,*
   - *otherwise $e_n \xrightarrow{\ell} e_{n+1}$ and $r_n \approx [\ell] \cdot r_{n+1}$ and $\widetilde{r}_n = [\ell] \cdot \widetilde{r}_{n+1}$*
2. *$e_n \rightsquigarrow \widetilde{r}_n$*
3. *if $r_n = (v_\infty, o^n_\infty)$, then $\widetilde{r}_n = (v_\infty, \widetilde{o}^n_\infty)$.*

Note that, by combining point 1 and point 3, we get that all the results in both $(r_n)_{n\in\mathbb{N}}$ and $(\widetilde{r}_n)_{n\in\mathbb{N}}$ have the same first component (final value or divergence), since they are related by an equivalence that on the first component is the equality.

In order to state the next lemma, we introduce the auxiliary judgment $e \xrightarrow{\widetilde{o}}_\star e'$, stating that e reduces to e' in finitely many steps producing the (finite) stream $\widetilde{o}$. The judgment is inductively defined as follows:

$$\frac{}{e \xrightarrow{[\,]}_\star e} \qquad \frac{e' \xrightarrow{\widetilde{o}}_\star e'' \quad e \xrightarrow{\ell} e'}{e \xrightarrow{[\ell]\cdot\widetilde{o}}_\star e''}$$

It is easy to check that $e \rightsquigarrow \widetilde{o} \cdot \widetilde{r}$ holds if and only if $e \xrightarrow{\widetilde{o}}_\star e'$ and $e' \rightsquigarrow \widetilde{r}$ for some e'.

▶ **Lemma 12.** *If $e \Rightarrow [v] \cdot r$, then there is a finite stream $\widetilde{o}$ such that $e \xrightarrow{\widetilde{o}}_\star \mathcal{E}[\texttt{out}\ v]$.*

Intuitively, the above lemma ensures that, if in the semantics by corules we produce an output, then in the LTS semantics we will reduce in finitely many steps to (an expression which contains) a corresponding output expression.

We are now ready to prove the first part of Theorem 7.

Proof. (Theorem 7 (1))

By Theorem 11 (points 1 and 2) we know that there are sequences $(e_n)_{n\in\mathbb{N}}$, $(r_n)_{n\in\mathbb{N}}$ and $(\widetilde{r}_n)_{n\in\mathbb{N}}$ such that $e_0 = e$, $r_0 = r$ and $e_0 \rightsquigarrow \widetilde{r}_0$. Therefore we have only to prove that $r_0 \approx \widetilde{r}_0$. To this aim, we prove by coinduction that, for all $n \in \mathbb{N}$, $r_n \approx \widetilde{r}_n$. By Theorem 11 (point 3) we already know that on the first component $\widetilde{r}_n$ and r_n are equal, hence we have only to reason on the stream part. Again by Theorem 11 (point 1) we also know that for all $n \in \mathbb{N}$, either $r_n \approx [\ell_n] \cdot r_{n+1}$ and $\widetilde{r}_n = [\ell_n] \cdot \widetilde{r}_{n+1}$, or $r_n = \widetilde{r}_n = (v, [\,])$. So, considering $r_n = (v_\infty, o^n_\infty)$ and $\widetilde{r}_n = (v_\infty, \widetilde{o}^n_\infty)$, we have three cases.

- If $\widetilde{r}_n = (v, [\,])$, then $e_n = v$, hence $\widetilde{r}_n = r_n = (v, [\,])$ by Theorem 11 (1), thus they are equivalent by axiom (TAU) in Fig. 7.
- If $\widetilde{r}_n = (v_\infty, \tau^\alpha)$ for $\alpha \in \mathbb{N} \cup \{\omega\}$ with $\alpha \neq 0$, then for all $n \leq k < n + \alpha$ we have that ℓ_k exists (by Theorem 11 (1) we have $e_k \xrightarrow{\ell_k} e_{k+1}$) and $\ell_k = \tau$. We have to prove that $r_n = (v_\infty, [\,])$. Let us assume that $r_n = [v] \cdot r'$, hence, since by Theorem 11 (1) $e_n \Rightarrow r_n$ holds, by Lemma 12 we get that there is a finite stream $\widetilde{o}$ such that $e_n \xrightarrow{\widetilde{o}}_\star \mathcal{E}[\mathtt{out}\ v]$ and by the determinism of $\xrightarrow{\ell}$ we get that there is $m \geq n$ such that $e_m = \mathcal{E}[\mathtt{out}\ v]$. Therefore we get that $e_m \xrightarrow{v} e_{m+1} = \mathcal{E}[v]$ (rules (OUT) and (CTX) in Fig. 1), hence $\ell_m = v \neq \tau$ that is a contradiction; thus $r_n = (v_\infty, [\,])$. Therefore we get the thesis by the first axiom.
- If $\widetilde{r}_n = \tau^k \cdot [v] \cdot \widetilde{r}_{n+k+1}$, then $\ell_{n+k} = v$ and for all $n \leq l < n + k$ we have $\ell_l = \tau$. Therefore we get by Theorem 11 (1) that $r_n \approx \tau^k \cdot [v] \cdot r_{n+k+1}$, hence $r_n = [v] \cdot r_{n+k+1}$. Finally by coinductive hypothesis we know that $r_{n+k+1} \approx \widetilde{r}_{n+k+1}$ ad thus we get the thesis by the following rule:

$$\frac{r_{n+k+1} \approx \widetilde{r}_{n+k+1}}{r_n \equiv [v] \cdot r_{n+k+1} \approx \tau^k \cdot [v] \cdot \widetilde{r}_{n+k+1} \equiv \widetilde{r}_n}$$

◀

In order to prove the point 2 of Theorem 7 we treat separately the cases of convergence and divergence. To prove the theorem we will show that if $e \rightsquigarrow \widetilde{r}$ holds, then $e \Rightarrow \varepsilon_\tau(\widetilde{r})$ holds too, where $\varepsilon_\tau(v_\infty, \widetilde{o}_\infty) = (v_\infty, \varepsilon_\tau(\widetilde{o}_\infty))$. This immediately proves the theorem, since $\varepsilon_\tau(\widetilde{r})$ is the witness we need to prove the existential quantification, and by definition of $\approx$ we have $\widetilde{r} \approx \varepsilon_\tau(\widetilde{r})$.

Denoting by $(\mathcal{I}, \mathcal{I}^{\mathsf{co}})$ the generalized inference system defining the judgment $e \Rightarrow r$ in Fig. 2, in the following we will call "extended system" the inference system $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$, that is, the (standard) inference system that is the union of rules and corules.

First of all, recall from the previous section conservativity for $e \rightsquigarrow \widetilde{r}$ (Theorem 5), that is, if $e \rightsquigarrow \widetilde{r}$ holds, then $\widetilde{r} = (v, \widetilde{o})$ if and only if $e \rightsquigarrow \widetilde{r}$ has a finite proof tree. This result allows us to reason by induction on rules defining convergence.

▶ **Lemma 13** (Subject expansion). *If $e \xrightarrow{\ell} e'$ and $e' \Rightarrow r'$ has a finite derivation in the extended system, then $e \Rightarrow r$ has a finite derivation in the extended system and $r \approx [\ell] \cdot r'$.*

Proof. Straightforward induction on rules defining $\xrightarrow{\ell}$. ◀

▶ **Theorem 14.** *If $e \rightsquigarrow (v, \widetilde{o})$, then $e \Rightarrow (v, \varepsilon_\tau(\widetilde{o}))$.*

Proof. By induction on the rules defining $e \rightsquigarrow \widetilde{r}$.

- If $v \rightsquigarrow (v, [\,])$, then the thesis follows from rule (VAL).
- If $e \rightsquigarrow [\ell] \cdot \widetilde{r}$, $e \xrightarrow{\ell} e'$ and $e' \rightsquigarrow \widetilde{r}$, then by inductive hypothesis we get $e' \Rightarrow \varepsilon_\tau(\widetilde{r})$, hence by Lemma 13 we get $e \Rightarrow r$ with $r \approx [\ell] \cdot \varepsilon_\tau(\widetilde{r})$ and this implies, by definition of $\approx$, that $r = \varepsilon_\tau([\ell] \cdot \widetilde{r})$. ◀

Now let us consider the case of divergence.

▶ **Lemma 15.** *$\mathcal{E}[\mathtt{out}\ v] \Rightarrow (\infty, [v] \cdot o_\infty)$ has a finite derivation in the extended system.*

Proof. By induction on the definition of contexts.

$\square$ We get the thesis by rules (VAL) and (CO-OUT).

$\mathtt{out}\ \mathcal{E}[\mathtt{out}\ v]$ We get the thesis by inductive hypothesis and rule (DIV).

$\mathcal{E}[\mathtt{out}\ v]\ \ e$ We get the thesis by inductive hypothesis and rule (DIV).

$v'\ \mathcal{E}[\mathtt{out}\ v]$ We get the thesis by rule (VAL), inductive hypothesis and rule (DIV). ◀

▶ **Theorem 16.** *If $e \rightsquigarrow (\infty, \widetilde{o}_\infty)$, then $e \Rightarrow (\infty, \varepsilon_\tau(\widetilde{o}_\infty))$ has a finite derivation in the extended system.*

Proof. We have two cases:

- if $\widetilde{o}_\infty = \tau^\alpha$ with $\alpha \in \mathbb{N} \cup \{\omega\}$, then $\varepsilon_\tau(\widetilde{o}_\infty) = [\,]$, hence the thesis follows from the rule (CO-EMPTY);
- otherwise, we have $\widetilde{o}_\infty = \tau^n \cdot [v] \cdot \widetilde{o}'_\infty$, hence by definition of $e \rightsquigarrow \widetilde{r}$, we get that $e \xrightarrow{\tau^n}_\star \mathcal{E}[\texttt{out}\ v]$ and $\mathcal{E}[\texttt{out}\ v] \rightsquigarrow (\infty, [v] \cdot \widetilde{o}'_\infty)$; therefore, by Lemma 15 we get that $\mathcal{E}[\texttt{out}\ v] \Rightarrow (\infty, [v] \cdot \varepsilon_\tau(\widetilde{o}'_\infty))$ is derivable by a finite proof tree in the extended system, then by a transitive closure of Lemma 13 we get the thesis, since $\varepsilon_\tau(\widetilde{o}_\infty) = [v] \cdot \varepsilon_\tau(\widetilde{o}'_\infty)$. ◀

Proof. (Theorem 7 (2)) We show by bounded coinduction (Theorem 3) that if $e \rightsquigarrow \widetilde{r}$, then $e \Rightarrow \varepsilon_\tau(\widetilde{r})$. The boundedness condition immediately follows from Theorem 14, Theorem 4 and Theorem 16. Let us proceed with the coinductive step: if $\widetilde{r} = (v, \widetilde{o})$, then the thesis follows from Theorem 14, hence we assume that $\widetilde{r} = (\infty, \widetilde{o}_\infty)$ and proceed by case analysis on e.

v Empty case.

x Empty case.

$\texttt{out}\ e$ Since $\texttt{out}\ e \rightsquigarrow (\infty, \widetilde{o}_\infty)$ holds, we get that $e \rightsquigarrow (\infty, \widetilde{o}_\infty)$ holds, hence by coinductive hypothesis $e \Rightarrow (\infty, \varepsilon_\tau(\widetilde{o}_\infty))$ and by rule (DIV) we get the thesis.

$e_1\ \ e_2$ We have three cases:

- If $e_1 \rightsquigarrow (\infty, \widetilde{o}_\infty)$, then by coinductive hypothesis we get $e_1 \Rightarrow (\infty, \varepsilon_\tau(\widetilde{o}_\infty))$, hence we get the thesis by rule (DIV).
- If $e_1 \rightsquigarrow (v, \widetilde{o})$ and $e_2 \rightsquigarrow (\infty, \widetilde{o}'_\infty)$, then $\widetilde{o}_\infty = \widetilde{o} \cdot \widetilde{o}'_\infty$, by Theorem 14 and coinductive hypothesis we get that $e_1 \Rightarrow (v, \varepsilon_\tau(\widetilde{o}))$ and $e_2 \Rightarrow (\infty, \varepsilon_\tau(\widetilde{o}'_\infty))$, hence we get the thesis by rule (DIV) since $\varepsilon_\tau(\widetilde{o}_\infty) = \varepsilon_\tau(\widetilde{o}) \cdot \varepsilon_\tau(\widetilde{o}'_\infty)$.
- If $e_1 \rightsquigarrow (\lambda x.e, \widetilde{o}_1)$ and $e_2 \rightsquigarrow (v_2, \widetilde{o}_2)$, then we have that $e_1\ \ e_2 \xrightarrow{\widetilde{o}_1}_\star \lambda x.e\ \ e_2 \xrightarrow{\widetilde{o}_2}_\star \lambda x.e\ \ v_2 \xrightarrow{\tau} e[x \leftarrow v_2] = e'$ with $e' \rightsquigarrow (\infty, \widetilde{o}'_\infty)$, and $\widetilde{o}_\infty = \widetilde{o}_1 \cdot \widetilde{o}_2 \cdot \widetilde{o}'_\infty$. Therefore by Theorem 14 and coinductive hypothesis we get $e_1 \Rightarrow (\lambda x.e, \varepsilon_\tau(\widetilde{o}_1))$, $e_2 \Rightarrow (v_2, \varepsilon_\tau(\widetilde{o}_2))$ and $e' \Rightarrow (\infty, \varepsilon_\tau(\widetilde{o}'_\infty))$, hence we get the thesis by rule (APP) since we have $\varepsilon_\tau(\widetilde{o}_\infty) = \varepsilon_\tau(\widetilde{o}_1) \cdot \varepsilon_\tau(\widetilde{o}_2) \cdot \varepsilon_\tau(\widetilde{o}'_\infty)$. ◀

6 A simple imperative Java-like language

In this section we provide the semantics by corules of an imperative Java-like language with `in` and `out` statements for reading and writing values, respectively. Our motivations for studying this language are the following.

- To check the approach on a (slightly) more realistic example than in Sect. 3. Notably, the language considered in this section is imperative and allows update of both local variables (in this simple calculus, just method parameters) and object fields, hence, stack frames and the heap need to be modeled. Moreover, since object values are heap references, they are read and written in a serialized format, and the serialization function needs to be coinductively defined.
- To investigate input as well. Managing both input and output requires deeper insights, since several approaches are possible; the one we propose, where I/O operations are treated as "events", seems to be simpler and is easily extensible to other kinds of significant interactions of the program with the outside, as acquisition and release of resources.

$$
\begin{array}{rcl}
p & ::= & \overline{cd}\ e \\
cd & ::= & \texttt{class}\ c_1\ \texttt{extends}\ c_2\ \{\ \overline{fd}\ \overline{md}\ \} \\
fd & ::= & f; \\
md & ::= & m(\overline{x})\ \{e\} \\
e & ::= & \texttt{new}\ c(\overline{e}) \mid x \mid \texttt{false} \mid \texttt{true} \mid e.f \mid e_0.m(\overline{e}) \mid x = e \mid e_1.f = e_2 \mid \texttt{in} \mid \texttt{out}\ e \\
 & & \mid \texttt{if}\ (e)\ e_1\ \texttt{else}\ e_2
\end{array}
$$

Figure 8 Java-like language: syntax.

$$
\begin{array}{rcll}
v, u & ::= & \texttt{false} \mid \texttt{true} \mid \iota & \text{(internal) value} \\
\mathrm{v} & ::= & \texttt{false} \mid \texttt{true} \mid obj(c, \overline{f}^n \mapsto \overline{\mathrm{v}}^n) & \text{serialized value} \\
\mathrm{e} & ::= & \texttt{in}\,\mathrm{v} \mid \texttt{out}\,\mathrm{v} & \text{event} \\
\theta & ::= & [\mathrm{e}_1 \ldots \mathrm{e}_n] \mid [\mathrm{e}_1 \ldots \mathrm{e}_n \ldots] & \text{event trace} \\
r & ::= & (v, \theta); \Pi; \mathcal{H} \mid (\infty, \theta) & \text{result} \\
\mathcal{D}[\,] & ::= & \texttt{new}\ c(\overline{\square}^n, \overline{e}) \mid \square.f \mid \square.m(\overline{\square}^k, \overline{e}) \quad (n > 0,\ k \geq 0) & \text{propagation context} \\
 & & \mid x = \square \mid \square.f = e \mid \square.f = \square \mid \texttt{out}\ \square \mid \texttt{if}\ (\square)\ e_1\ \texttt{else}\ e_2 &
\end{array}
$$

Figure 9 Java-like language: values, results and propagation contexts.

The syntax of the language is defined in Fig. 8. We write $\overline{cd}^n$, or simply $\overline{cd}$, for the sequence $cd_1 \ldots cd_n$, and analogously for other sequences. With abused notation $\overline{f}^n \mapsto \overline{v}^n$ stands for $f_1 \mapsto v_1, \ldots, f_n \mapsto v_n$. We assume sets of *variables* (*parameters*) x, including `this`, *class names* c, including `Object`, *field names* f, and *method names* m.

The calculus is an imperative untyped Java-like language, where variables and fields can be updated; since the work is focused on the dynamic semantics, we have simplified the language by omitting type annotations.

For simplicity, statements are expressions with side effects; assignments to variables and fields return the value of the right-hand side expression, the `in` statement returns the deserialized value that has been acquired from the input, while the `out` e statement returns the value obtained from the evaluation of e, which is also output.

We assume standard syntactic restrictions: inheritance is acyclic, names are distinct in class, method, field, and parameter declarations, `Object` cannot be declared, class names other than `Object` that are referred in expressions must be declared.

A program consists of a sequence of class declarations and a main expression; class declarations contain field and method definitions, whereas a single constructor is implicitly defined, with parameters corresponding to the inherited and defined fields, with precedence to the inherited ones. In a method body the target object is accessed via the implicit read-only parameter `this`, which is assumed to differ from all explicitly declared parameters. Statement expressions include instance creation, variable, boolean literals, field selection, method invocation, variable and field update, input/output operations, and conditional expression.

We define now the semantics by corules of the language; as for the λ-calculus in Sect. 3, we provide a semantics with the fully deterministic left-to-right call-by-value evaluation strategy. Internal and serialized values, events, event traces, results and (divergence) propagation contexts are defined in Fig. 9. As for the λ-calculus, propagation contexts can contain many holes, all at fixed depth 1, and allow a more compact definition of propagation of diverging results (see rule (DIV) in Fig. 10). We assume an infinite set of *object references* ι. Internal

values are either boolean constants or object references; for boolean values the serialized form is the same, whereas object references are serialized to values of shape $obj(c, \overline{f}^n \mapsto \overline{\mathrm{v}}^n)$, with c the class of the object, and $\overline{f}^n$ its fields associated with their corresponding serialized values $\overline{\mathrm{v}}^n$; serialized objects are allowed to be regular terms, to allow serialization of cyclic objects.

Events that are tracked by the semantics are input/output operations, hence, they have shape `in` v (input of serialized value v) and `out` v (output of serialized value v). Traces are finite or infinite sequences of tracked events.

Results of converging computations are triples $(v, \theta); \Pi; \mathcal{H}$ where the first component is a pair (v, θ) consisting of a returned value, and a produced finite event trace, while the second and third ones are the stack frame Π and the heap $\mathcal{H}$ yielded by the computation, respectively. Results model also diverging computations with pairs of shape (∞, θ), where the event trace θ is allowed to be infinite; in case of diverging computations, neither returned value is defined, nor stack frame and heap are yielded. The stack frame of the method under execution is a finite partial map Π from variables (`this` and the explicit parameters of the method) to values. The heap $\mathcal{H}$ is a finite partial map from references to objects. Objects are pairs $obj(c, \rho)$, where c is the object's class and ρ is a finite partial map from field names to values. We use the usual set-theoretic representation for finite maps, in particular for heaps $\mathcal{H}$, maps ρ from fields to values, and maps from fields to serialized values in serialized objects; the operator $\uplus$ denotes union of maps with disjoint domains. We assume that heaps cannot contain dangling references: for all references ι, if $\mathcal{H}(\iota) = obj(c, \rho)$, then for all $f \in dom(\rho)$, $\rho(f) \in dom(\mathcal{H}) \uplus \{\mathtt{false}, \mathtt{true}\}$; indeed, in the language object references cannot be explicitly deallocated.

The semantics is formalized by the judgment $\Pi; \mathcal{H}; e \Rightarrow r$, where r is either $(v, \theta); \Pi'; \mathcal{H}'$ or (∞, θ). In the former case, the judgment means that, in stack frame Π and heap $\mathcal{H}$, expression e converges to value v, with finitely generated event trace θ, and yielded stack frame Π' and heap $\mathcal{H}'$; in the latter, expression e diverges with possibly infinite event trace θ.

The semantic rules are defined in Fig. 10; for brevity, we leave implicit the index of the judgment that should consist of the class declarations contained in the program.

The definitions of all auxiliary functions used in the side conditions of the rules can be found in the Appendix.

As in Fig. 2, rule (DIV) propagates diverging computations: if the evaluation of a subexpression diverges before all remaining subexpressions are evaluated, then the control flow of the program has to be modified; this happens in all situations captured by the propagation contexts defined in Fig. 9.

Rules (VAR) and (BOOL) are straightforward: the computation always converges and returns an empty trace.

Rules (NEW) and (FLD) are standard; in (NEW) the generated trace is obtained by concatenating in the same order the traces returned by the evaluation of the argument expressions; the disjoint union $\uplus$ ensures that ι is a fresh reference in the current heap $\mathcal{H}_n$, the side condition requires that the arguments match all fields inherited and declared by class c. In (FLD) the generated trace corresponds to that obtained from the evaluation of the subexpression denoting the target object; the side condition ensures that such an evaluation returns an object reference defined in the heap and containing the accessed field.

Similarly to rule (APP) for function application in Fig. 2, rule (INV) deals with method invocation when the evaluation of the target and all arguments converges; the auxiliary function *restore* (see the comments to Fig. 12 in the Appendix) is used for restoring the current stack frame Π_{n+1} of the caller yielded after the evaluation of the target and the

$$\text{(DIV)}\ \frac{\forall\, i = 1..n.\Pi_{i-1}; \mathcal{H}_{i-1}; e_i \Rightarrow (v_i, \theta_i); \Pi_i; \mathcal{H}_i \quad \Pi_n; \mathcal{H}_n; e \Rightarrow (\infty, \theta)}{\Pi_0; \mathcal{H}_0; \mathcal{D}[\overline{e}^n, e] \Rightarrow (\infty, \theta_1 \cdot \ldots \cdot \theta_n \cdot \theta)}$$

$$\text{(VAR)}\ \frac{}{\Pi; \mathcal{H}; x \Rightarrow (v, [\,]); \Pi; \mathcal{H}}\ \Pi(x) = v \qquad \text{(BOOL)}\ \frac{}{\Pi; \mathcal{H}; e \Rightarrow (e, [\,]); \Pi; \mathcal{H}}\ e \in \{\texttt{false}, \texttt{true}\}$$

$$\text{(NEW)}\ \frac{\forall\, i = 1..n.\Pi_{i-1}; \mathcal{H}_{i-1}; e_i \Rightarrow (v_i, \theta_i); \Pi_i; \mathcal{H}_i}{\Pi_0; \mathcal{H}_0; \texttt{new}\ c(\overline{e}^n) \Rightarrow (\iota, \theta_1 \cdot \ldots \cdot \theta_n); \Pi_n; \mathcal{H}_n \uplus \{\iota \mapsto obj(c, \overline{f}^n \mapsto \overline{v}^n)\}}\ fields(c) = \overline{f}^n$$

$$\text{(FLD)}\ \frac{\Pi_0; \mathcal{H}_0; e \Rightarrow (\iota, \theta); \Pi_1; \mathcal{H}_1}{\Pi_0; \mathcal{H}_0; e.f \Rightarrow (v, \theta); \Pi_1; \mathcal{H}_1}\ \mathcal{H}_1(\iota) = obj(c, \rho \uplus \{f \mapsto v\})$$

$$\text{(INV)}\ \frac{\begin{array}{c}\forall\, i = 0..n.\Pi_i; \mathcal{H}_i; e_i \Rightarrow (v_i, \theta_i); \Pi_{i+1}; \mathcal{H}_{i+1} \\ \{\texttt{this} \mapsto v_0, \overline{x}^n \mapsto \overline{v}^n\}; \mathcal{H}_{n+1}; e \Rightarrow r\end{array}}{\Pi_0; \mathcal{H}_0; e_0.m(\overline{e}) \Rightarrow r'}\ \begin{array}{l}\mathcal{H}_1(v_0) = obj(c, \rho) \\ meth(c, m) = \overline{x}^n.e \\ r' = restore(\Pi_{n+1}, (\theta_0 \cdot \ldots \cdot \theta_n) \cdot r)\end{array}$$

$$\text{(VAS)}\ \frac{\Pi_0; \mathcal{H}_0; e \Rightarrow (v, \theta); \Pi_1 \uplus \{x \mapsto u\}; \mathcal{H}_1}{\Pi_0; \mathcal{H}_0; x = e \Rightarrow (v, \theta); \Pi_1 \uplus \{x \mapsto v\}; \mathcal{H}_1}$$

$$\text{(FAS)}\ \frac{\Pi_0; \mathcal{H}_0; e_1 \Rightarrow (\iota, \theta_1); \Pi_1; \mathcal{H}_1 \quad \Pi_1; \mathcal{H}_1; e_2 \Rightarrow (v, \theta_2); \Pi_2; \mathcal{H} \uplus \{\iota \mapsto obj(c, \rho \uplus \{f \mapsto u\})\}}{\Pi_0; \mathcal{H}_0; e_1.f = e_2 \Rightarrow (v, \theta_1 \cdot \theta_2); \Pi_2; \mathcal{H} \uplus \{\iota \mapsto obj(c, \rho \uplus \{f \mapsto v\})\}}$$

$$\text{(IN)}\ \frac{}{\Pi; \mathcal{H}_0; \texttt{in} \Rightarrow (v, [\texttt{in}\,\mathrm{v}]); \Pi; \mathcal{H}_1}\ deser(\mathcal{H}_0, \mathrm{v}) = (\mathcal{H}_1, v)$$

$$\text{(OUT)}\ \frac{\Pi_0; \mathcal{H}_0; e \Rightarrow (v, \theta); \Pi_1; \mathcal{H}_1}{\Pi_0; \mathcal{H}_0; \texttt{out}\ e \Rightarrow (v, \theta \cdot [\texttt{out}\,\mathrm{v}]); \Pi_1; \mathcal{H}_1}\ ser(\mathcal{H}_1, v) = \mathrm{v}$$

$$\text{(IF)}\ \frac{\Pi_0; \mathcal{H}_0; e \Rightarrow (v, \theta); \Pi_1; \mathcal{H}_1 \quad \Pi_1; \mathcal{H}_1; e_i \Rightarrow r}{\Pi_0; \mathcal{H}_0; \texttt{if}\ (e)\ e_1\ \texttt{else}\ e_2 \Rightarrow \theta \cdot r}\ v = \texttt{true} \wedge i = 1 \vee v = \texttt{false} \wedge i = 2$$

$$\text{(CO-EMPTY)}\ \frac{}{\Pi; \mathcal{H}; e \Rightarrow (\infty, [\,])} \qquad \text{(CO-IN)}\ \frac{}{\Pi; \mathcal{H}; \texttt{in} \Rightarrow (\infty, [\texttt{in}\,\mathrm{v}] \cdot \theta)}$$

$$\text{(CO-OUT)}\ \frac{\Pi; \mathcal{H}; e \Rightarrow (v, \theta'); \Pi'; \mathcal{H}'}{\Pi; \mathcal{H}; \texttt{out}\ e \Rightarrow (\infty, \theta' \cdot [\texttt{out}\,\mathrm{v}] \cdot \theta)}\ \mathrm{v} = ser(\mathcal{H}', v)$$

Figure 10 Java-like language: inference system with corules.

arguments of the method invocation. The body of the method is evaluated in the new stack frame $\{\texttt{this} \mapsto v_0, \overline{x}^n \mapsto \overline{v}^n\}$ defining `this`, and the formal parameters; its result can correspond to either a converging or a diverging computation. In the latter case, r has shape (∞, θ), with θ possibly infinite, the stack frame is not restored, but divergence is simply propagated to the conclusion of the rule, after concatenating to the left of θ, in the same evaluation order, the finite traces obtained from the evaluation of the target and the arguments; hence $restore(\Pi_{n+1}, (\theta_0 \cdot \ldots \cdot \theta_n) \cdot (\infty, \theta)) = (\infty, \theta_0 \cdot \ldots \cdot \theta_n \cdot \theta)$. The other side conditions are standard and ensure that the target of the invocation is an object whose class has method m with n parameters.

Rules (VAS) and (FAS) manage variable and field updates, respectively; the former changes[5] the stack frame, the latter the heap. Rule (VAS) is applicable only when the variable to update is defined in the current stack frame; the trace generated from the assignment corresponds to that obtained from the evaluation of the right-hand side subexpression. Rule (FAS) is applicable only when the target evaluates to a reference to an object containing the field to be updated; the trace generated from the assignment corresponds to the concatenation, in the same order, of the traces obtained from the evaluation of the target and right-hand side subexpression.

In rule (IN) a trace with the single event `in` v is generated, where v is allowed to be any possible serialized value that can be obtained from the input. The corresponding returned inner value v is the deserialization of v (see the comments to Fig. 12 in the Appendix); such an operation can extend the heap if v corresponds to an object. In this case a new internal object has to be allocated.

In rule (OUT) the trace is obtained by concatenating that returned by the evaluation of the subexpression with the singleton trace containing the event `out` v, where v is the serialization of the value v (see the comments to Fig. 12 in the Appendix) to be output.

Rule (IF) is standard; it is applicable only if the evaluation of the condition converges to a boolean value. However, the overall result is allowed to diverge; this happens when the evaluation of the selected branch does not terminate. In any case, the generated trace is obtained by concatenating the traces, in the same evaluation order, yielded by the evaluation of the condition and the selected branch.

As for the semantics of the λ-calculus in Sect. 3, corules filter out undesired behavior in case of non termination: results can only have shape (∞, θ), with θ possibly infinite; diverging computations which do not involve input/output operations give rise to proof trees where the definition of the yielded trace is non productive, hence corule (CO-EMPTY) forces the returned trace to be empty. The remaining corules (IN) and (OUT) relax the constraint of corule (CO-EMPTY) by allowing traces of arbitrary length (including infinity) when infinitely many input or output operations are performed, respectively; indeed, in both corules no constraint is imposed on the rest of the yielded trace, represented by the metavariable θ. As in Fig. 2, the premise of corule (CO-OUT) ensures that the evaluation of the subexpression denoting the value to output converges, to ensure that the corresponding serialized value is actually output.

We consider now an example program (other two examples are provided in the Appendix).

Example 1

Let us consider the program consisting of the following class declaration

```
class C extends Object{m(x){this.m(out in)}}
```

and the main expression $e =$ `new C().m(true)`; such a program diverges and produces an infinite trace with alternating input and output events s.t. each event `in` v is immediately followed by the event `out` v; this is formalized by the derivable judgment $\emptyset;\emptyset;e \Rightarrow (\infty, \theta_0)$, where $\theta_0 = [\texttt{in}\,\mathrm{v}_0\ \texttt{out}\,\mathrm{v}_0\ \texttt{in}\,\mathrm{v}_1\ \texttt{out}\,\mathrm{v}_1 \ldots]$.

Fig. 11 shows how an infinite tree can be derived with the standard rules of Fig. 10; for simplicity the figure considers only the cases where input values are just primitive boolean

[5] Of course, stack frame and heap can also be indirectly changed by the evaluation of the subexpressions of the statements.

$$\nabla_i = \text{(INV)}\frac{\text{(VAR)}\overline{\Pi_i;\mathcal{H};\texttt{this} \Rightarrow (\iota,[\,]);\Pi_i;\mathcal{H}} \quad \text{(OUT)}\frac{\text{(IN)}\overline{\Pi_i;\mathcal{H};\texttt{in} \Rightarrow (\mathrm{v}_i,[\texttt{in}\,\mathrm{v}_i]);\Pi_i;\mathcal{H}}}{\Pi_i;\mathcal{H};\texttt{out in} \Rightarrow (\mathrm{v}_i,[\texttt{in}\,\mathrm{v}_i\ \texttt{out}\,\mathrm{v}_i]);\Pi_i;\mathcal{H}} \quad \nabla_{i+1}}{\Pi_i;\mathcal{H};\texttt{this.m(out in)} \Rightarrow (\infty,[\texttt{in}\,\mathrm{v}_i\ \texttt{out}\,\mathrm{v}_i]\cdot\theta_{i+1})}$$

$$\text{(INV)}\frac{\text{(NEW)}\overline{\emptyset;\emptyset;\texttt{new C()} \Rightarrow (\iota,[\,]);\emptyset;\mathcal{H}} \quad \text{(BOOL)}\overline{\emptyset;\mathcal{H};\texttt{true} \Rightarrow (\texttt{true},[\,]);\emptyset;\mathcal{H}} \quad \nabla_0}{\emptyset;\emptyset;\texttt{new C().m(true)} \Rightarrow (\infty,\theta_0)}$$

where $\forall\, i \in \mathbb{N}.\theta_i = [\texttt{in}\,\mathrm{v}_i\ \texttt{out}\,\mathrm{v}_i]\cdot\theta_{i+1}, \mathcal{H} = \{\iota \mapsto \mathit{obj}(\mathtt{C},\emptyset)\}, \Pi_0 = \{\texttt{this} \mapsto \iota, \mathtt{x} \mapsto \texttt{true}\}$, $\Pi_{i+1} = \{\texttt{this} \mapsto \iota, \mathtt{x} \mapsto \mathrm{v}_i\}, \mathrm{v}_i \in \{\texttt{false},\texttt{true}\}$

Figure 11 Infinite derivation for $\emptyset;\emptyset;\texttt{new C().m(true)} \Rightarrow (\infty,\theta_0)$ in Example 1.

values and, hence, no heap memory is allocated when deserialization occurs every time a new value is read from the input through the `in` instruction. Hence, after the allocation triggered by the execution of `new C()`, the heap $\mathcal{H}$ remains unchanged; in case infinite serialized objects are read from the input, the heaps in the derived judgments of the proof tree grow indefinitely with the depth of the tree.

Although in Fig. 11 only primitive input values are considered, in general the derived infinite proof tree is not regular and consists of an infinite set of subtrees $\nabla_0, \nabla_1, \ldots$, each one depending from the particular input values $\mathrm{v}_0, \mathrm{v}_1, \ldots$; since at each call a value is input and then output, the definition for the trace θ_0 in the finally derived judgment is fully productive and no trace other than $[\texttt{in}\,\mathrm{v}_0\,\texttt{out}\,\mathrm{v}_0\ \texttt{in}\,\mathrm{v}_1\ \texttt{out}\,\mathrm{v}_1\ldots]$ can be derived; since such a trace is infinite, it is not possible to derive judgments of shape $\emptyset;\emptyset;\texttt{new C().m(true)} \Rightarrow (v,\theta_0);\emptyset;\mathcal{H}'$.

For each occurrence of rule (INV) in the infinite proof tree, only finite trees can be built for the evaluation of the target and argument of the method invocation, therefore for the corresponding premises only judgments for converging computations can be derived, and rule (DIV) cannot be applied in place of rule (INV).

Finally, to show that the proof tree in Fig. 11 is valid, we need to prove that by using also the corules we can derive finite proof trees for all judgments of the proof. To this aim, we can prove that for all judgments $\Pi_i;\mathcal{H};\texttt{this.m(out in)} \Rightarrow (\infty,[\texttt{in}\,\mathrm{v}_i\ \texttt{out}\,\mathrm{v}_i]\cdot\theta_{i+1})$ derived with ∇_i, it is possible to build the following finite tree:

$$\text{(DIV)}\frac{\text{(VAR)}\overline{\Pi_i;\mathcal{H};\texttt{this} \Rightarrow (\iota,[\,]);\Pi_i;\mathcal{H}} \quad \textbf{(co-out)}\frac{\text{(IN)}\overline{\Pi_i;\mathcal{H};\texttt{in} \Rightarrow (\mathrm{v}_i,[\texttt{in}\,\mathrm{v}_i]);\Pi_i;\mathcal{H}}}{\Pi_i;\mathcal{H};\texttt{out in} \Rightarrow (\infty,[\texttt{in}\,\mathrm{v}_i\ \texttt{out}\,\mathrm{v}_i]\cdot\theta_{i+1})}}{\Pi_i;\mathcal{H};\texttt{this.m(out in)} \Rightarrow (\infty,[\texttt{in}\,\mathrm{v}_i\ \texttt{out}\,\mathrm{v}_i]\cdot\theta_{i+1})}$$

Additional examples can be found in the Appendix.

We illustrate now by a simple example how to use the bounded coinduction technique (Theorem 3) to reason about concrete programs. Consider the following Java-like program:

```
class T extends Object{hasNext(){true}}
class F extends Object{hasNext(){false}}
class Main extends Object {
  loop(){if(in.hasNext()) this.loop() else false}
}
```

and abbreviate by `F` and `T`, respectively, the two input events $\texttt{in}\,\mathit{obj}(\mathtt{F},\emptyset)$ and $\texttt{in}\,\mathit{obj}(\mathtt{T},\emptyset)$.

Intuitively, there are two possible valid results of the program e =`new Main().loop()`. If the input provides infinitely many `T`'s, e loops forever, producing the corresponding infinite trace $[\mathtt{T}\ \ldots\mathtt{T}\ \ldots]$, abbreviated $\mathtt{T}^\infty$ in the following. If, after n `T`'s, an `F` is eventually

read, then the program terminates returning `false`, and producing a finite trace $[\mathtt{T}\ \ldots\ \mathtt{T}\ \mathtt{F}]$, abbreviated $\mathtt{T}^n$ in the following. Note that, if any other kind of (serialized) value is read, then the program execution is stuck, that is, there is no (either infinite or finite) result. In our formalization, differently from what happens in standard big-step semantics, this case is nicely distinguished from divergence, since in the former case there is no proof tree.

More precisely, valid judgments for e have one of the following shapes:

1. (a) $\emptyset;\emptyset;e \Rightarrow (\infty,\mathtt{T}^\infty)$
(b) $\emptyset;\emptyset;e \Rightarrow (\mathtt{false},\mathtt{T}^n);\emptyset;\{\iota \mapsto \mathit{obj}(\mathtt{Main},\emptyset)\} \uplus \mathcal{H}^n$

with $\mathcal{H}^n = \{\iota_i \mapsto \mathit{obj}(\mathtt{T},\emptyset) \mid i \in 1..n\} \uplus \{\iota' \mapsto \mathit{obj}(\mathtt{F},\emptyset)\}$.

In order to formally prove that such judgments are derivable, as it is customary in (co)induction proofs, we have to extend the set of valid judgments, including all those which are needed for the coinductive hypothesis:

2. $\emptyset;\emptyset;\mathtt{new\ Main()} \Rightarrow (\iota,[\,]);\emptyset;\{\iota \mapsto \mathit{obj}(\mathtt{Main},\emptyset)\}$
3. (a) $\Pi;\mathcal{H};\mathtt{if(in.hasNext())\ this.loop()\ else\ false} \Rightarrow (\infty,\mathtt{T}^\infty)$
(b) $\Pi;\mathcal{H};\mathtt{if(in.hasNext())\ this.loop()\ else\ false} \Rightarrow (\mathtt{false},\mathtt{T}^n);\Pi;\mathcal{H} \uplus \mathcal{H}^n$
4. (a) $\Pi;\mathcal{H};\mathtt{in.hasNext()} \Rightarrow (\mathtt{true},[\mathtt{T}]);\Pi;\mathcal{H} \uplus \{\iota \mapsto \mathit{obj}(\mathtt{T},\emptyset)\}$
(b) $\Pi;\mathcal{H};\mathtt{in.hasNext()} \Rightarrow (\mathtt{false},[\mathtt{F}]);\Pi;\mathcal{H} \uplus \{\iota \mapsto \mathit{obj}(\mathtt{F},\emptyset)\}$
5. (a) $\Pi;\mathcal{H};\mathtt{in} \Rightarrow (\iota,[\mathtt{T}]);\Pi;\mathcal{H} \uplus \{\iota \mapsto \mathit{obj}(\mathtt{T},\emptyset)\}$
(b) $\Pi;\mathcal{H};\mathtt{in} \Rightarrow (\iota,[\mathtt{F}]);\Pi;\mathcal{H} \uplus \{\iota \mapsto \mathit{obj}(\mathtt{F},\emptyset)\}$
6. (a) $\Pi;\mathcal{H};\mathtt{true} \Rightarrow (\mathtt{true},[\,]);\Pi;\mathcal{H}$ (b) $\Pi;\mathcal{H};\mathtt{false} \Rightarrow (\mathtt{false},[\,]);\Pi;\mathcal{H}$
7. $\Pi;\mathcal{H};\mathtt{this} \Rightarrow \iota;\Pi;\mathcal{H}$
8. (a) $\Pi;\mathcal{H};\mathtt{this.loop()} \Rightarrow (\infty,\mathtt{T}^\infty)$
(b) $\Pi;\mathcal{H};\mathtt{this.loop()} \Rightarrow (\mathtt{false},\mathtt{T}^n);\Pi;\mathcal{H} \uplus \mathcal{H}^n$

where $\Pi = \{\mathtt{this} \mapsto \iota\}$, $\mathcal{H}(\iota) = \mathit{obj}(\mathtt{Main},\emptyset)$.

Set $\mathcal{S}$ the set of judgments of shape 1-8, $(\mathcal{I},\mathcal{I}^{\mathsf{co}})$ the generalized inference system defining the judgment $\Pi;\mathcal{H};e \Rightarrow r$ in Fig. 10, and $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$ the (standard) inference system that is the union of $\mathcal{I}$ and $\mathcal{I}^{\mathsf{co}}$. To prove by bounded coinduction (Theorem 3) that each judgment in $\mathcal{S}$ can be derived, we have to show:

Boundedness. Each judgment in $\mathcal{S}$ has a finite proof tree in $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$.

For convergent judgments (all cases except 1a, 3a, 8a) it is easy to show that they have a finite proof tree in $\mathcal{I}$, hence in $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$ as well. In particular, for cases 3b, 8b this finite proof tree can be constructed by arithmetic induction on n.

For cases 1a, 3a, 8a it is enough to show that a finite proof tree for $\Pi;\mathcal{H};\mathtt{in} \Rightarrow (\infty,\mathtt{T}^\infty)$ in $\mathcal{I} \cup \mathcal{I}^{\mathsf{co}}$ can be obtained by corule (CO-IN), analogously to what we have shown for Example 3 in Sect. 3.

Coinductive step. Each judgment in $\mathcal{S}$ is the consequence of an inference rule in $\mathcal{I}$ where all premises are in $\mathcal{S}$.

The proof can be done by cases. For instance, for case 3b, the judgment is the consequence of rule (IF) with first premise of shape 4 and second premise of shape either 8b or 6b.

We conclude this section with a comment on the proof technique outlined above. Here we have added in the set $\mathcal{S}$ of valid judgments exactly those strictly needed for the coinductive hypothesis. This is a minimal choice. A different approach, which we used in [6] under the name *divergence consistency principle*, is to add to $\mathcal{S}$ all the judgments which are derivable in $\mathcal{I}$. With this approach, it is enough to prove conservativity (the analogous of Theorem 4), and that each valid diverging judgment (cases 1a, 3a, 8a in our example) is the consequence of a rule where all diverging premises are valid as well, and all converging premises are derivable. This technique makes the proof schema simpler, but here we preferred the explicit approach for illustrating better the specific example.

7 Related work

One of the first approaches to model divergence with operational semantic rules was proposed by Cousot and Cousot [8] for the call-by-value λ-calculus by means of two different judgments defined in a stratified way: the former with inductive rules to model termination, the latter with coinductive rules and depending from the former, to capture divergence. In a followup work [9] they proposed a more sophisticated framework based on bi-inductive definitions, an order-theoretic approach to inductive definitions which allows the simultaneous definition of finite and infinite behaviors in operational semantics; however, such a solution has been adopted only for the standard call-by-value λ-calculus, and no uses of it can be found in literature for expressing the semantics of other languages.

Leroy and Grall [13] have investigated several operational semantics of the call-by-value λ-calculus for capturing divergence, their equivalence, and suitability to formally prove type soundness and compiler correctness; they are all defined in terms of inductive and coinductive judgments defined in a stratified way.

Coinductive big-step semantics has been proposed by Ancona to proof soundness for a Java-like language [3].

An operational semantics modeling divergence with an ad hoc coinductive judgment has been investigated also by Charguéraud with the notion [7] of pretty-big step semantics.

Flag-based big-step semantics [18] is a recent approach able to capture divergence by interpreting the same semantic rules both inductively and coinductively; flags represent termination or divergence and are part of the result of the computation. In case of non termination, values (or other semantic details exclusively used for terminating computations) are non deterministically returned.

We are planning to investigate the relationship between corules and conditional coinduction [10] employed by Danielsson to combine induction and coinduction in definitions of total parser combinators. Followup work [11], inspired by the paper of Leroy and Grall [13], shows how the use the coinductive partiality monad allow the definition of big/small-step semantics for lambda-calculi and virtual machines as total, computable functions able to capture divergence.

Several approaches to divergence with operational semantics [17, 2, 4] have been inspired by the notion of definitional interpreter [19]. All semantic definitions depend on a counter which limits the number of steps a computation can take; if the value of the counter is not sufficient to complete the computation, then a timeout is returned. In this way divergence can be modeled by induction.

Owens et al. [17] investigate functional big-step semantics for proving by induction compiler correction, including divergence preservation. Amin and Rompf [2] explore inductive proof strategies for type soundness properties for the polymorphic type systems $F_{<:}$, and equivalence with small-step semantics. Ancona [4] proposes an inductive proof of type soundness for the big-step semantics of a Java-like language.

More recently, Ancona et al. [6] have shown that with coaxioms it is possible to mix together induction and coinduction in a single inference system to define a big-step operational semantics able to capture divergence with just one judgment. The call-by-value λ-calculus is considered, and a proof of equivalence with the standard small-step semantics is provided. Then the semantics of an imperative Java-like language is defined, and a corresponding type soundness result is proved.

All papers surveyed so far limit their investigation to semantics which capture divergence, but, differently to the contribution of this paper, do not provide any support for reasoning

on the behavior of non terminating programs, because in case of non termination the only information that is conveyed is divergence.

Nakata and Uustalu [14, 15, 16] have investigated on coinductive trace semantics in big-step style; they started with the semantics of an imperative While language with no I/O [14] where traces are possibly infinite sequences of states; semantic rules are all coinductive and define two mutually dependent judgments. Based on such a semantics, they define a Hoare logic [15]; differently to our approach, weak trace equivalence is required for proving that programs exhibit equivalent observable behaviors. A constructive theory and metatheory and a Coq formalization are provided.

The semantics has been subsequently extended with interactive I/O [16], by exploiting the notion of resumption: a tree representing possible runs of a program to model its non-deterministic behavior due to input values. Also in this case a big-step trace semantics is defined with two mutually recursive coinductive judgments, and weak bisimilarity is needed; however, the definition of the observational equivalence is more involved, since it requires nesting inductive definitions in coinductive ones.

In both papers [14, 15] equivalence of the big-step and small-step semantics is proved; also, in the considered languages [14, 15] expressions and statements are distinct, and expressions cannot diverge (divergence is not obtained through infinite recursion, but rather through infinite while loops). This is a significant difference with the languages we have considered in this paper; under the assumption that the evaluation of e cannot diverge, the semantics of `out` e becomes simpler, indeed, the corresponding corule could be turned into a coaxiom.

8 Conclusion

We have shown how generalized inference systems can be employed for formalizing in a convenient way non trivial operational semantics suitable for reasoning on the behavior of possibly diverging programs.

The two examples of semantics we have provided suggest that by using corules with a similar pattern, other notions of interesting infinite behaviour can be modeled to reason on properties of diverging programs, to prove, for instance, that a server program uses a finite amount of system resources, even when expected to never stop.

As a byproduct, we have extended the theory of generalized inference systems with the more general notion of corule, for which significant examples were missing until now.

We briefly discuss now advantages, drawbacks, and limitations of the approach. While we do not claim that our approach is *easier* than the standard formulation using labeled small-step semantics, an important advantage is that the whole system is directly based on a unique judgment, thus allowing more direct reasoning (proofs).

A difficulty in adopting the approach could be how to define the right corules, which requires new expertise in comparison with the well established inductive reasoning (essentially based on case analysis). For this reason, we are currently working on the definition of "canonical" corule patterns. Notably, the two examples in this paper should become instances of a general transformation from an inductive inference system modeling finite behaviour to an inference system with corules modeling infinite behaviour as well.

Of course, inference systems with corules are not a silver bullet for expressing any kind of recursive definition. The main limitation we have found until now is in modeling, roughly, recursive functions where the result can be an infinite set. For instance, the inference system with corules for the graph example at page 4 is not complete when the set of nodes is infinite. A similar example (the infinite carrier of a list) is mentioned in [5]. We plan to investigate and possibly address this limitation.

For the first simpler example of semantics we have fully proved that our approach is equivalent to a semantics based on the standard notion of LTS and observational equivalence; we leave for future work a proof of equivalence for the imperative Java-like language. Although in this case the technical details are more complex, we do not expect any surprise in the proofs of equivalence.

Besides the already mentioned directions, it would be of great interest to try to test our approach with the support of a proof assistant.

References

1. Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical logic*, pages 739–782. North Holland, 1977.
2. Nada Amin and Tiark Rompf. Type soundness proofs with definitional interpreters. In Giuseppe Castagna and Andrew D. Gordon, editors, *ACM Symp. on Principles of Programming Languages 2017*, pages 666–679. ACM Press, 2017. doi:10.1145/3009837.
3. Davide Ancona. Soundness of object-oriented languages with coinductive big-step semantics. In James Noble, editor, *ECOOP'12 - Object-Oriented Programming*, volume 7313 of *Lecture Notes in Computer Science*, pages 459–483. Springer, 2012. doi:10.1007/978-3-642-31057-7_21.
4. Davide Ancona. How to prove type soundness of Java-like languages without forgoing big-step semantics. In David J. Pearce, editor, *FTfJP'14 - Formal Techniques for Java-like Programs*, pages 1:1–1:6. ACM Press, 2014. doi:10.1145/2635631.2635846.
5. Davide Ancona, Francesco Dagnino, and Elena Zucca. Generalizing inference systems by coaxioms. In Hongseok Yang, editor, *ESOP 2017 - European Symposium on Programming*, volume 10201 of *Lecture Notes in Computer Science*, pages 29–55. Springer, 2017. doi:10.1007/978-3-662-54434-1_2.
6. Davide Ancona, Francesco Dagnino, and Elena Zucca. Reasoning on divergent computations with coaxioms. *PACMPL*, 1(OOPSLA):81:1–81:26, 2017. doi:10.1145/3133905.
7. Arthur Charguéraud. Pretty-big-step semantics. In Matthias Felleisen and Philippa Gardner, editors, *ESOP 2013 - European Symposium on Programming*, volume 7792 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2013. doi:10.1007/978-3-642-37036-6_3.
8. Patrick Cousot and Radhia Cousot. Inductive definitions, semantics and abstract interpretations. In Ravi Sethi, editor, *ACM Symp. on Principles of Programming Languages 1992*, pages 83–94. ACM Press, 1992. doi:10.1145/143165.143184.
9. Patrick Cousot and Radhia Cousot. Bi-inductive structural semantics. *Information and Computation*, 207(2):258–283, 2009. doi:10.1016/j.ic.2008.03.025.
10. Nils Anders Danielsson. Total parser combinators. In *Intl. Conf. on Functional Programming 2010*, pages 285–296. ACM Press, 2010.
11. Nils Anders Danielsson. Operational semantics using the partiality monad. In Peter Thiemann and Robby Bruce Findler, editors, *Intl. Conf. on Functional Programming 2012*, pages 127–138. ACM Press, 2012. doi:10.1145/2364527.2364546.
12. Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016. doi:10.1017/CBO9781316823187.
13. Xavier Leroy and Hervé Grall. Coinductive big-step operational semantics. *Information and Computation*, 207(2):284–304, 2009. doi:10.1016/j.ic.2007.12.004.
14. Keiko Nakata and Tarmo Uustalu. Trace-based coinductive operational semantics for while. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *The-*

orem Proving in Higher Order Logics - TPHOLs 2009, volume 5674 of *Lecture Notes in Computer Science*, pages 375–390. Springer, 2009. doi:10.1007/978-3-642-03359-9_26.

15 Keiko Nakata and Tarmo Uustalu. A Hoare logic for the coinductive trace-based big-step semantics of while. In Andrew D. Gordon, editor, *ESOP 2010 - European Symposium on Programming*, volume 6012 of *Lecture Notes in Computer Science*, pages 488–506. Springer, 2010. doi:10.1007/978-3-642-11957-6_26.

16 Keiko Nakata and Tarmo Uustalu. Resumptions, weak bisimilarity and big-step semantics for while with interactive I/O: an exercise in mixed induction-coinduction. In Luca Aceto and Pawel Sobocinski, editors, *SOS'10 - Structural Operational Semantics*, volume 32 of *Electronic Proceedings in Theoretical Computer Science*, pages 57–75, 2010. doi:10.4204/EPTCS.32.5.

17 Scott Owens, Magnus O. Myreen, Ramana Kumar, and Yong Kiam Tan. Functional big-step semantics. In Peter Thiemann, editor, *ESOP 2016 - European Symposium on Programming*, volume 9632 of *Lecture Notes in Computer Science*, pages 589–615. Springer, 2016. doi:10.1007/978-3-662-49498-1_23.

18 C. B. Poulsen and P. D. Mosses. Flag-based big-step semantics. *Journal of Logical and Algebraic Methods in Programming*, 2016.

19 John C. Reynolds. Definitional interpreters for higher-order programming languages. In *ACM '72, Proceedings of the ACM annual conference*, volume 2, pages 717—-740. ACM Press, 1972.

20 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

A A coalgebraic view

We recall some basic notions about coalgebras, see for instance [12, 20]. Given a category $\mathcal{C}$ and an endofunctor $F\colon \mathcal{C} \to \mathcal{C}$, an *F-coalgebra* is a pair (C, γ) where C is an object of $\mathcal{C}$ and $\gamma\colon C \to FC$ is an arrow in $\mathcal{C}$. An *F-coalgebra homomorphism* between two F-coalgebras (C, γ) and (C', γ') is an arrow $f\colon C \to C'$ in $\mathcal{C}$ such that $\gamma' \cdot f = Ff \cdot \gamma$, where $\cdot$ denotes the arrow composition in $\mathcal{C}$. It is easy to check that the composition of coalgebra homomorphisms is again an homomorphism and that the identity arrow on the carrier of an F-coalgebra is an homomorphism, hence F-coalgebras and homomorphisms form a category and the terminal object in this category, if any, is named *terminal F-coalgebra*.

We now fix $\mathcal{C} = \mathsf{Set}$, that is, we work in the category of sets and functions. Given two F-coalgebras (C, γ) and (C', γ'), a *bisimulation* between them is a relation $R \subseteq C \times C'$ that carries an F-coalgebra structure such that the canonical projections $\pi_1\colon R \to C$ and $\pi_2\colon R \to C'$ are F-coalgebra homomorphisms. In other words, a bisimulation is a relation that agrees with the coalgebraic structure of its components.

Let us now introduce some notations. If A is a set, A^∞ is the set of finite and infinite streams over A. We denote by $\mathcal{V}$ the set of values, and by $\mathcal{V}_\tau$ the set $\mathcal{V} + \{\tau\}$, where $+$ denotes the coproduct in the category Set of sets and functions.

It is well-known that $\mathcal{V}^\infty$ is the carrier of the terminal coalgebra of $F\colon \mathsf{Set} \to \mathsf{Set}$, the functor defined by $FX = \mathbf{1} + \mathcal{V} \times X$, with the map $\zeta\colon \mathcal{V}^\infty \to F\mathcal{V}^\infty$ defined by

$$\zeta(x) = \begin{cases} (v, x') & \text{if } x = [v] \cdot x' \\ \star & \text{otherwise } (x = [\,]) \end{cases}$$

where $\star \in \mathbf{1}$ is the unique element of the terminal object $\mathbf{1}$ in Set, However, we can give an

F-coalgebra structure also to $\mathcal{V}_\tau^\infty$, considering the function $\gamma_\tau \colon \mathcal{V}_\tau^\infty \to F\mathcal{V}_\tau^\infty$ defined by

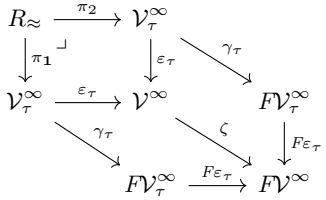
$$\gamma_\tau(x) = \begin{cases} (v, x') & \text{if } x = \tau^n \cdot [v] \cdot x' \\ \star & \text{otherwise } (x = \tau^\alpha) \end{cases}$$

Since $(\mathcal{V}^\infty, \zeta)$ is terminal, there is a unique F-coalgebra homomorphism $\varepsilon_\tau \colon \mathcal{V}_\tau^\infty \to \mathcal{V}^\infty$, in other words ε_τ is the unique map making the following diagram commute:

$$\begin{array}{ccc} \mathcal{V}_\tau^\infty & \xrightarrow{\varepsilon_\tau} & \mathcal{V}^\infty \\ \downarrow{\scriptstyle \gamma_\tau} & & \downarrow{\scriptstyle \zeta} \\ F\mathcal{V}_\tau^\infty & \xrightarrow{F\varepsilon_\tau} & F\mathcal{V}^\infty \end{array}$$

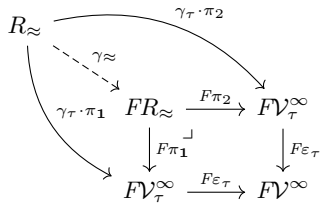
Intuitively, this diagram forces ε_τ to satisfy the equations mentioned in the beginning, hence to be the function that removes τs from a stream.

In order to construct $\approx$, we consider the set $R_\approx$ defined as the pullback in Set of the pair of functions $(\varepsilon_\tau, \varepsilon_\tau)$, hence, since ε_τ is an F-coalgebra homomorphism, we get the following commutative diagram:



More explicitly, $R_\approx$ is the set $\{(x, y) \in \mathcal{V}_\tau^\infty \times \mathcal{V}_\tau^\infty \mid \varepsilon_\tau(x) = \varepsilon_\tau(y)\}$, thus it is an *equivalence relation* on $\mathcal{V}_\tau^\infty$.

It is easy to check that F preserves pullbacks in Set, hence we get the following commutative diagram:



Therefore $(R_\approx, \gamma_\approx)$ is an F-coalgebra, π_1 and π_2 are F-coalgebra homomorphisms and hence $R_\approx$ is a *bisimulation equivalence* on $(\mathcal{V}_\tau^\infty, \gamma_\tau)$.

We now show that $\approx$ and $R_\approx$ are indeed the same relation.

▶ **Proposition 17.** *$\widetilde{o}_\infty \approx \widetilde{o}'_\infty$ if and only if $(\widetilde{o}_\infty, \widetilde{o}'_\infty) \in R_\approx$.*

B Proofs

Proof of Prop. 6 ($\approx$ is an equivalence) We show that $\approx$ is reflexive, symmetric and transitive.

Reflexivity We have to prove that, for each stream $\widetilde{o}_\infty$, $\widetilde{o}_\infty \approx \widetilde{o}_\infty$ holds; we proceed by coinduction. We distinguish two cases:

- if $\widetilde{o}_\infty = \tau^\alpha$ for some $\alpha \in \mathbb{N} \cup \{\omega\}$, then the thesis follows by the first rule, taking β equal to α;

- otherwise, there is a value v such that $\widetilde{o}_\infty = \tau^n \cdot [v] \cdot \widetilde{o}'_\infty$, hence by coinductive hypothesis we get $\widetilde{o}'_\infty \approx \widetilde{o}'_\infty$, and the thesis follows from the second rule, taking m equal to n.

Symmetry Assume $\widetilde{o}_\infty \approx \widetilde{o}'_\infty$, we have to prove that $\widetilde{o}'_\infty \approx \widetilde{o}_\infty$; we proceed by coinduction. We distinguish two cases:

- if both $\widetilde{o}_\infty$ and $\widetilde{o}'_\infty$ are made of only τs, then the thesis follows from the first rule (it is enough to swap the two exponents);
- otherwise, we have $\widetilde{o}_\infty = \tau^n \cdot [v] \cdot \widetilde{s}_\infty$ and $\widetilde{o}'_\infty = \tau^m \cdot [v] \cdot \widetilde{s}'_\infty$ with $\widetilde{s}_\infty \approx \widetilde{s}'_\infty$; hence by coinductive hypothesis we get $\widetilde{s}'_\infty \approx \widetilde{s}_\infty$, and the thesis follows by applying the rule

$$\frac{\widetilde{s}'_\infty \approx \widetilde{s}_\infty}{\widetilde{o}'_\infty = \tau^m \cdot [v] \cdot \widetilde{s}'_\infty \approx \tau^n \cdot [v] \cdot \widetilde{s}_\infty = \widetilde{o}_\infty}$$

Transitivity Let us assume $\widetilde{o}_\infty \approx \widetilde{o}'_\infty$ and $\widetilde{o}'_\infty \approx \widetilde{o}''_\infty$. We proceed by coinduction. We distinguish two cases:

- if $\widetilde{o}'_\infty = \tau^\alpha$ for some $\alpha \in \mathbb{N} \cup \{\omega\}$, then also $\widetilde{o}_\infty$ and $\widetilde{o}''_\infty$ are made of only τs, hence the thesis follows from the first rule;
- otherwise, we have $\widetilde{o}_\infty = \tau^p \cdot [v] \cdot \widetilde{s}_\infty$, $\widetilde{o}'_\infty = \tau^q \cdot [v] \cdot \widetilde{s}'_\infty$ and $\widetilde{o}''_\infty = \tau^r \cdot [v] \cdot \widetilde{s}''_\infty$, with $\widetilde{s}_\infty \approx \widetilde{s}'_\infty$ and $\widetilde{s}'_\infty \approx \widetilde{s}''_\infty$. Therefore by coinductive hypothesis we get $\widetilde{s}_\infty \approx \widetilde{s}''_\infty$, and the thesis follows by applying the rule

$$\frac{\widetilde{s}_\infty \approx \widetilde{s}''_\infty}{\widetilde{o}_\infty = \tau^p \cdot [v] \cdot \widetilde{s}_\infty \approx \tau^r \cdot [v] \cdot \widetilde{s}''_\infty = \widetilde{o}''_\infty}$$

Proof of Prop. 17 ($\approx$ and $R_\approx$ coincide) For the implication $\Leftarrow$ we proceed by coinduction, considering two cases:

- if $\varepsilon_\tau(\widetilde{o}_\infty) = \varepsilon_\tau(\widetilde{o}'_\infty) = [\,]$, then $\widetilde{o}_\infty = \tau^\alpha$ and $\widetilde{o}'_\infty = \tau^\beta$, hence we get the thesis by the first rule;
- otherwise we have $\widetilde{o}_\infty = \tau^n \cdot [v] \cdot \widetilde{s}_\infty$, $\widetilde{o}'_\infty = \tau^m \cdot [v] \cdot \widetilde{s}'_\infty$ and $\varepsilon_\tau(\widetilde{s}_\infty) = \varepsilon_\tau(\widetilde{s}'_\infty)$. Hence by coinductive hypothesis we get $\widetilde{s}_\infty \approx \widetilde{s}'_\infty$ and then we get the thesis by the second rule.

To show the other implication, we have to prove that the set $E = \{(\varepsilon_\tau(\widetilde{o}_\infty), \varepsilon_\tau(\widetilde{o}'_\infty)) \mid \widetilde{o}_\infty \approx \widetilde{o}'_\infty\}$ is included in the diagonal relation on $\mathcal{V}^\infty$. To this aim, thanks to the (coalgebraic) coinduction principle, it is enough to show that E is a bisimulation, that is, that E is an F-coalgebra. Consider the following function defined on E:

$$\begin{cases} \gamma_E([\,],[\,]) & = \star \\ \gamma_E([v] \cdot \varepsilon_\tau(\widetilde{s}_\infty), [v] \cdot \varepsilon_\tau(\widetilde{s}'_\infty)) & = (v, (\varepsilon_\tau(\widetilde{s}_\infty), \varepsilon_\tau(\widetilde{s}'_\infty))) \end{cases}$$

and note that by definition of $\approx$ we have $\widetilde{s}_\infty \approx \widetilde{s}'_\infty$, hence $(\varepsilon_\tau(\widetilde{s}_\infty), \varepsilon_\tau(\widetilde{s}'_\infty)) \in E$, and so $\gamma_E \colon E \to FE$ is well-defined, and (E, γ_E) is an F-coalgebra, and this concludes the proof.

Proof of Lemma 12 We relax the hypothesis, only requiring that $e \Rightarrow [v] \cdot r$ is derivable by a finite proof tree using also corules. Indeed, by definition, if $e \Rightarrow [v] \cdot r$ is derivable, then it has a finite proof tree using also corules. We proceed by induction on the definition.

(co-empty) Empty case.

(co-out) We know that $\mathtt{out}\, e \Rightarrow [v] \cdot o_\infty$ and $e \Rightarrow (v', o)$. We have two cases:

- if $o = [v] \cdot o'$, then, by inductive hypothesis, $e \xrightarrow{\widetilde{o}}_\star \mathcal{E}[\mathtt{out}\, v]$, hence $\mathtt{out}\, e \xrightarrow{\widetilde{o}}_\star \mathtt{out}\, \mathcal{E}[\mathtt{out}\, v]$;
- if $o = [\,]$, then $v' = v$ and by Lemma 10 we get that $e \rightsquigarrow (v, \widetilde{o})$, hence $e \xrightarrow{\widetilde{o}}_\star v$; hence we get that $\mathtt{out}\, e \xrightarrow{\widetilde{o}}_\star \mathtt{out}\, v$ as needed.

(val) Empty case.

(out) Analogous to case (CO-OUT).

(app) We know that $e_1\ e_2 \Rightarrow [v]\cdot r$, $e_1 \Rightarrow (\lambda x.e, o_1)$, $e_2 \Rightarrow (v_2, o_2)$ and $e[x \leftarrow v_2] \Rightarrow r'$. We distinguish three cases:

- if $o_1 = [v]\cdot o_1'$, then by inductive hypothesis we get that $e_1 \xrightarrow{\widetilde{o}}_\star \mathcal{E}[\mathtt{out}\ v]$, hence $e_1\ e_2 \xrightarrow{\widetilde{o}}_\star \mathcal{E}[\mathtt{out}\ v]\ e_2$ as needed;
- if $o_1 = [\,]$ and $o_2 = [v]\cdot o_2'$, then by Lemma 10 $e_1 \rightsquigarrow (\lambda x.e, \widetilde{o}_1)$, hence $e_1 \xrightarrow{\widetilde{o}_1}_\star \lambda x.e$, and by inductive hypothesis $e_2 \xrightarrow{\widetilde{o}}_\star \mathcal{E}[\mathtt{out}\ v]$, therefore we get that $e_1\ e_2 \xrightarrow{\widetilde{o}_1}_\star \lambda x.e\ e_2 \xrightarrow{\widetilde{o}}_\star \lambda x.e\ \mathcal{E}[\mathtt{out}\ v]$ as needed;
- if $o_1 = o_2 = [\,]$, then $r' = [v]\cdot r$, hence by Lemma 10 we get that $e_1 \rightsquigarrow (\lambda x.e, \widetilde{o}_1)$ and $e_2 \rightsquigarrow (v_2, \widetilde{o}_2)$, hence $e_1 \xrightarrow{\widetilde{o}_1}_\star \lambda x.e$ and $e_2 \xrightarrow{\widetilde{o}_2}_\star v_2$, and by inductive hypothesis $e[x \leftarrow v_2] \xrightarrow{\widetilde{o}}_\star \mathcal{E}[\mathtt{out}\ v]$; therefore we get that $e_1\ e_2 \xrightarrow{\widetilde{o}_1}_\star \lambda x.e\ e_2 \xrightarrow{\widetilde{o}_2}_\star \lambda x.e\ v_2 \xrightarrow{\tau} e[x \leftarrow v_2] \xrightarrow{\widetilde{o}}_\star \mathcal{E}[\mathtt{out}\ v]$ as needed.

(div) Analogous to case (APP).

Proof of Theorem 11 We prove separately the three points.

1. We start by constructing inductively the sequences $(e_n)_{n\in\mathbb{N}}$ and $(r_n)_{n\in\mathbb{N}}$. The case $n = 0$ is given by the hypothesis. If we assume e_n and r_n to satisfy $e_n \Rightarrow r_n$, then we have two cases:

- if $e_n = v$, then we set $e_{n+1} = v$ and $r_{n+1} = (v, [\,])$;
- otherwise, by Lemma 8 there are e' and ℓ such that $e_n \xrightarrow{\ell} e'$, and by Lemma 9 there is r' such that $e' \Rightarrow r'$ and $r_n \approx [\ell]\cdot r'$; therefore we set $e_{n+1} = e'$ and $r_{n+1} = r'$.

In this way, by construction $(e_n)_{n\in\mathbb{N}}$ and $(r_n)_{n\in\mathbb{N}}$ satisfy the requirements.

In order to construct the sequence $(\widetilde{r}_n)_{n\in\mathbb{N}}$ we need some more effort. First of all, we have to find an infinite sequence of streams satisfying the requirements of point 1, that is, a function $s\colon \mathbb{N} \to \mathcal{V}_\tau^\infty$ such that, for all $n \in \mathbb{N}$, if e_n is a value, then $s(n) = [\,]$, and if $e_n \xrightarrow{\ell} e_{n+1}$, then $s(n) = [\ell]\cdot s(n+1)$. To do this, we give a coalgebra structure on $\mathbb{N}$ for the functor $FX = \mathbf{1} + \mathcal{V}_\tau \times X$, given by the map $\gamma\colon \mathbb{N} \to F\mathbb{N}$ defined by

$$\gamma(n) = \begin{cases} \star & e_n = v \\ (\ell, n+1) & e_n \xrightarrow{\ell} e_{n+1} \end{cases}$$

Now, since $\mathcal{V}_\tau^\infty$ carries a terminal F-coalgebra, and requirements of point 1 impose that s is a homomorphism, it is uniquely determined. Therefore we set $\widetilde{r}_n = (v, s(n))$ if there is $k \geq n$ such that $e_k = v$, otherwise $\widetilde{r}_n = (\infty, s(n))$.

2. We proceed by coinduction, distinguishing two cases:

- if e_n is a value v, then, by point 1, $\widetilde{r}_n = (v, [\,])$, hence $e_n \rightsquigarrow \widetilde{r}_n$ holds by the first axiom;
- otherwise, by point 1, $e_n \xrightarrow{\ell} e_{n+1}$, $e_{n+1} \Rightarrow r_{n+1}$ and $\widetilde{r}_n = [\ell]\cdot \widetilde{r}_{n+1}$, therefore by coinductive hypothesis we get $e_{n+1} \rightsquigarrow \widetilde{r}_{n+1}$ and so by the second rule we get the thesis.

3. We distinguish two cases:

- if $v_\infty = v$, then, since for all $n \in \mathbb{N}$ by points 1 and 2 we have $e_n \Rightarrow (v, o_n)$ and $e_n \rightsquigarrow \widetilde{r}_n$, by Lemma 10 we get the thesis;
- consider a result r_n in the sequence, and assume $v_\infty = \infty$, then for all $k \geq n$ we have $e_k \neq v$, since otherwise we would get $v_\infty = v$, because $r_n \approx o\cdot r_k$, by point 1; therefore by construction (point 1) we get $\widetilde{r}_n = (\infty, \widetilde{o}_\infty^n)$.

$$\frac{}{\mathit{fields}(\texttt{Object}) = \epsilon} \qquad \frac{\begin{array}{c}\mathit{fields}(c') = \overline{f} \\ \texttt{class } c \texttt{ extends } c' \ \{\ \overline{g}; \overline{md}\} \\ \overline{f} \cap \overline{g} = \emptyset\end{array}}{\mathit{fields}(c) = \overline{f}, \overline{g}}$$

$$\frac{\texttt{class } c \texttt{ extends } c' \ \{\ \overline{fd}\ \ \overline{md}\ \ m(\overline{x})\ \{e\}\ \overline{md'}\}}{\mathit{meth}(c, m) = \overline{x}.e} \qquad \frac{\begin{array}{c}\mathit{meth}(c', m) = \overline{x}.e \\ \texttt{class } c \texttt{ extends } c' \ \{\ \overline{fd}\ \ \overline{md}\} \\ m \text{ not declared in } \overline{md}\end{array}}{\mathit{meth}(c, m) = \overline{x}.e}$$

$$\mathit{restore}(\Pi, r) = \begin{cases} r & \text{if } r = (\infty, \theta) \\ (v, \theta); \Pi; \mathcal{H} & \text{if } r = (v, \theta); \Pi'; \mathcal{H} \end{cases} \qquad \theta \cdot ((v, \theta'); \Pi; \mathcal{H}) = (v, \theta \cdot \theta'); \Pi; \mathcal{H}$$

$$\theta \cdot (\infty, \theta') = (\infty, \theta \cdot \theta')$$

$$\frac{}{\mathit{ser}(\mathcal{H}, v) = v}\ v \in \{\texttt{false}, \texttt{true}\} \qquad \frac{\forall\, i = 1..n. \mathit{ser}(\mathcal{H}, v_i) = \mathrm{v}_i}{\mathit{ser}(\mathcal{H}, \iota) = \mathit{obj}(c, \overline{f}^n \mapsto \overline{\mathrm{v}}^n)}\ \mathcal{H}(\iota) = \mathit{obj}(c, \overline{f}^n \mapsto \overline{v}^n)$$

$$\frac{}{\mathit{ser}(\mathcal{H}, v) = \mathrm{v}}$$

$$\frac{\mathit{deser}(\mathcal{H}, \mathrm{v}, \emptyset) = (\mathcal{H}', v)}{\mathit{deser}(\mathcal{H}, \mathrm{v}) = (\mathcal{H}', v)} \qquad \frac{}{\mathit{deser}(\mathcal{H}, \mathrm{v}, M) = (\mathcal{H}, \mathrm{v})}\ \mathrm{v} \in \{\texttt{false}, \texttt{true}\}$$

$$\frac{}{\mathit{deser}(\mathcal{H}, \mathrm{v}, M \uplus \{\mathrm{v} \mapsto \iota\}) = (\mathcal{H}, \iota)}$$

$$\frac{\forall\, i = 1..n. \mathit{deser}(\mathcal{H}_{i-1}, \mathrm{v}_i, M \cup \{\mathrm{v} \mapsto \iota\}) = (\mathcal{H}_i, v_i)}{\mathit{deser}(\mathcal{H}_0, \mathrm{v}, M) = (\mathcal{H}_n \cup \{\iota \mapsto \mathit{obj}(c, \overline{f}^n \mapsto \overline{v}^n)\}, \iota)} \quad \begin{array}{l}\mathrm{v} \notin \mathit{dom}(M) \\ \mathrm{v} = \mathit{obj}(c, \overline{f}^n \mapsto \overline{\mathrm{v}}^n) \\ \iota \notin \mathit{dom}(\mathcal{H}_0) \cup \mathit{img}(M)\end{array}$$

Figure 12 Definition of auxiliary functions and operators

C Auxiliary definitions and additional examples for the imperative Java-like language

Auxiliary definitions. Fig. 12 shows the definition of all auxiliary functions and operators used in the rules.

Functions *fields* and *meth* are standard. Function *restore* replaces the stack frame of the callee with that of the caller after the computation returns from a method invocation (rule (INV)).

The operation $\theta \cdot r$ updates the result r by appending the finite trace θ to the left of the possibly infinite trace θ' of r.

Function *ser* corresponds to built-in serialization of an internal value v into a corresponding serialized value v; because object values are heap references, the function depends also on the heap. Serialization of primitive values is trivial, and serialization of objects does not yield a sequence (as happens in practice), but rather preserves the tree structure of the original object, allowed to be infinite, but still regular because heaps have finite domains. For this reason, we provide a standard coinductive definition of *ser*.

The inverse function *deser* depends on heaps as well, since deserialization of objects requires object allocation; for the same reason, the function returns a pair consisting of a new heap and a value. Deserialization of cyclic objects requires particular care, because one has to avoid infinite unfolding which would lead to a heap with an infinite domain; our choice is to

minimize unfolding, therefore recursive deserialization stops as soon as a loop is detected in the structure of the serialized object, and no redundant cycles are introduced in the heap. To this aim, the definition of *deser* is based on an overloaded function *deser* taking a third argument M, which is a finite map from serialized objects to their associated reference, to keep track of cycles. If a serialized object v is already in the domain of M, then a cycle has been detected, and the associated reference in M and the unchanged heap are returned. Otherwise, deserialization is propagated to the fields of v with an updated map M where a fresh reference ι is associated with v. Finally, an updated heap and the new reference ι are returned. Because of the use of the third argument M, the definition of $deser(\mathcal{H}, \mathrm{v}, M)$ is inductive.

Example 2. This example is a simple variation of Example 1, where method `m` does not output the input value.

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

In this case the only derivable judgments have shape $\emptyset;\emptyset;e \Rightarrow (\infty, \theta_0)$, where the event trace θ_0 is defined by $\theta_0 = [\mathtt{in}\,\mathrm{v}_0\,\mathtt{in}\,\mathrm{v}_1\ldots]$. The corresponding infinite proof is obtained from that of Example 1 by slightly changing the trees ∇_i (with the same simplifying assumption that only primitive input values are considered); in this case we have

$$\nabla_i = \text{(INV)}\frac{\text{(VAR)}\overline{\Pi_i;\mathcal{H};\mathtt{this} \Rightarrow (\iota, [\,]);\Pi_i;\mathcal{H}} \quad \text{(IN)}\overline{\Pi_i;\mathcal{H};\mathtt{in} \Rightarrow (\mathrm{v}_i, [\mathtt{in}\,\mathrm{v}_i]);\Pi_i;\mathcal{H}} \quad \nabla_{i+1}}{\Pi_i;\mathcal{H};\mathtt{this.m(in)} \Rightarrow (\infty, [\mathtt{in}\,\mathrm{v}_i]\cdot\theta_{i+1})}$$

where for all $i \in \mathbb{N}$, $\theta_i = [\mathtt{in}\,\mathrm{v}_i]\cdot\theta_{i+1}$, whereas $\mathcal{H}$, and Π_i are defined as before. The following finite trees built with corules show that the infinite trees ∇_i are valid:

$$\text{(DIV)}\frac{\text{(VAR)}\overline{\Pi_i;\mathcal{H};\mathtt{this} \Rightarrow (\iota, [\,]);\Pi_i;\mathcal{H}} \quad \text{(CO-IN)}\overline{\Pi_i;\mathcal{H};\mathtt{in} \Rightarrow (\infty, [\mathtt{in}\,\mathrm{v}_i]\cdot\theta_{i+1});\Pi_i;\mathcal{H}}}{\Pi_i;\mathcal{H};\mathtt{this.m(in)} \Rightarrow (\infty, [\mathtt{in}\,\mathrm{v}_i]\cdot\theta_{i+1})}$$

Example 3. This example is a more elaborated variation of Example 1, where some computation is performed on input values before they are output, and a simple cache object is employed. To simplify the code we employ the primitive operator `==`, together with integer literals and addition.

```
class H extends Object{ // simple cache objects
   // fields store the last input and its associated output
   input; output;
   get(i){
      if(i==this.input) this.output
      else this.output=this.calc(this.input=i)
   }
   /* performs some computation on i and returns the result */
   calc(i){ i+1 }
}
class C extends Object{cache; m(x){this.m(out cache.get(in))}}
```

If we consider the main expression $e =$ `new C(new H(0,0)).m(true)`, and restrict the observation to positive integers as input, then the only derivable judgments have shape $\emptyset;\emptyset;e \Rightarrow (\infty, \theta_0)$, where $\theta_0 = [\mathtt{in}\,\mathrm{v}_0 \quad \mathtt{out}\,\mathrm{v}_0 + 1 \quad \mathtt{in}\,\mathrm{v}_1 \quad \mathtt{out}\,\mathrm{v}_1 + 1\ldots]$, with v_i a positive integer for all $i \in \mathbb{N}$.