A meta-theory for big-step semantics

FRANCESCO DAGNINO, Università di Genova, Italy

It is well-known that big-step semantics is not able to distinguish stuck and non-terminating computations. This is a strong limitation as it makes very difficult to reason about properties involving infinite computations, such as type soundness, which cannot even be expressed.

We show that this issue is only apparent: the distinction between stuck and diverging computations is implicit in any big-step semantics and it just needs to be uncovered. To achieve this goal, we develop a systematic study of big-step semantics: we introduce an abstract definition of what a big-step semantics is, we define a notion of computation by formalising the evaluation algorithm implicitly associated with any big-step semantics, and we show how to canonically extend a big-step semantics to characterise stuck and diverging computations.

Building on these notions, we describe a general proof technique to show that a predicate is sound, that is, it prevents stuck computation, with respect to a big-step semantics. One needs to check three properties relating the predicate and the semantics and, if they hold, the predicate is sound. The extended semantics are essential to establish this meta-logical result, but are of no concerns to the user, who only needs to prove the three properties of the initial big-step semantics. Finally, we illustrate the technique by several examples, showing that it is applicable also in cases where subject reduction does not hold, hence the standard technique for small-step semantics cannot be used.

CCS Concepts: • **Theory of computation → Operational semantics**.

Additional Key Words and Phrases: big-step semantics, type soundness

ACM Reference Format:
Francesco Dagnino. 2020. A meta-theory for big-step semantics. *ACM Trans. Comput. Logic* 37, 4, Article 111 (August 2020), 50 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The operational semantics of programming languages or software systems specifies, for each program/system configuration, its final result, if any. In the case of non-existence of a final result, there are two possibilities:

- either the computation stops with no final result: *stuck computation*,
- or the computation never stops: *non-termination*.

There are two main styles to define operationally a semantic relation: the *small-step* style [46, 47], on top of a transition relation representing single computation steps, or directly by a set of rules as in the *big-step* style [35]. Within a small-step semantics it is straightforward to make the distinction between stuck and non-terminating computations, while a typical drawback of the big-step style is that they are not distinguished (no judgement is derived in both cases).

Author's address: Francesco Dagnino, francesco.dagnino@dibris.unige.it, DIBRIS, Università di Genova, Genova, Italy.

Manuscript submitted to ACM

Actually, in big-step style, it is not even clear *what a computation is*, because the only available notion is derivability of judgements, which does not convey the dynamics of computation.

For this reason, even though big-step semantics is generally more abstract, and sometimes more intuitive to design and therefore to debug and extend, in the literature much more effort has been devoted to study the meta-theory of small-step semantics, providing properties, and related proof techniques. Notably, the *soundness* of a type system (typing prevents stuck computation) can be proved by *progress* and *subject reduction*, also called *type preservation*, [53]. Note that soundness cannot even be *expressed* with respect to a big-step semantics, since non-termination and stuckness are confused, as they are both modelled by the absence of a final result.

Our quest in this paper is to develop a meta-theory of big-step operational semantics, to enable formal reasoning also on non-terminating computations. More precisely, we will address the following problems:

(1) Defining, in a formal way, *computations* in a given arbitrary big-step semantics.
(2) According to this definition, describing *extensions of a given arbitrary big-step semantics*, where the difference between stuckness and non-termination is made explicit.
(3) Providing a general proof technique by identifying *three sufficient conditions* on the original big-step rules to prove soundness of a predicate.

All these points rely on the same fundamental cornerstone: a general definition of big-step semantics. Such a definition captures the essential features of a big-step semantics, independently from the particular language or system.

To address Item 1, we rely on the intuition that every big-step semantics implicitly defines an evaluation algorithm. Then, we identify computations in the big-step semantics with computations of such algorithm. Formally, we extend the big-step semantics to model *partial evaluations*, representing intermediate states of the evaluation process, and we formalise the evaluation algorithm by a transition relation between such intermediate states. Then, computations are just sequences of transition steps. Note that the use of a transition relation is somehow necessary to define computations since they are related to the dynamics of the evaluation and it cannot be captured by derivability in big-step semantics, as it is too abstract. In this way, we get a reference model of computations in big-step semantics, where we can easily distinguish stuck and non-terminating computations, thus showing that this distinction is actually present, but hidden, in any big-step semantics.

To deal with Item 2, we describe extensions of a given big-step semantics capable to distinguish between stuck and non-terminating computations, as defined in Item 1, but abstracting away single computation steps. In this way, we show that such a distinction can be made directly in a big-step style. More in detail, starting from an arbitrary big-step judgment $c \Rightarrow r$ that evaluates *configurations* c into *results* r, the first construction produces an enriched judgement $c \Rightarrow_{\mathrm{tr}} r_{\mathrm{tr}}$ where r_{tr} is either a pair $\langle t, r \rangle$ consisting of a finite trace t and a result r, or an infinite trace σ. Finite and infinite traces model the (finite or infinite) sequences of all the configurations encountered during the evaluation. In this way, by interpreting coinductively the rules of the extended semantics, an infinite trace models divergence (whereas no result corresponds to stuck computation). Furthermore, we will show that, by using *coaxioms* [8, 22], we can get rid of traces, modelling divergence just by a judgmeent $c \Rightarrow \infty$. The second construction is in a sense dual. It is the general version of the well-known technique presented in Exercise 3.5.16 by Pierce [44] of adding a special result wrong explicitly modelling stuck computations (whereas no result corresponds to divergence). We will show that these constructions are correct, proving that they represent the intended class of computations as defined in Item 1.

Three sufficient conditions in Item 3 are *local preservation*, $\exists$*-progress*, and $\forall$*-progress*. For *proving* the result that the three conditions actually ensure soundness, we crucially rely on the extended big-step semantics of Item 2, since otherwise, as said above, we could not even express the property.

However, the three conditions deal only with the original rules of the given big-step semantics. This means that, practically, in order to use the technique there is no need to deal with the meta-theory (computations and extended semantics). This implies, in particular, that our approach does *not* increase the original number of rules. Moreover, the sufficient conditions are checked only on *single rules*, hence neither induction nor coinduction is needed. In a sense, they make explicit elementary fragments of the soundness proof, embedding such semantic-dependent fragments in a semantic-independent (co)inductive proof, which we carry out once and for all (cf. Theorems 7.6 and 7.9).

We support our approach by presenting several examples, demonstrating that: on the one hand, soundness proofs can be easily rephrased in terms of our technique, that is, by directly reasoning on big-step rules; on the other hand, our technique works also when the property to be checked (for instance, well-typedness) is *not preserved* by intermediate computation steps, whereas it holds for the final result. On a side note, our examples concern type systems, but the meta-theory we present in this work holds for any predicate.

Actually, we can express two flavours of soundness, depending on whether we make explicit stuckness or non-termination. In the former case we express *soundness-must*, which is the notion of soundness we have considered so far, preventing all stuck computations, while in the latter case we express *soundness-may*, a weaker notion only ensuring *the existence of* a non-stuck computation. Of course, this distinction is relevant only in presence of non-determinism, otherwise the two notions coincide. We define a proof technique for soundness-may as well, showing it is correct. In the end, it should be noted that *we define soundness with respect to a big-step semantics within a big-step formulation*, without resorting to a small-step style (indeed, the extended semantics are themselves big-step).

This paper is extracted from the PhD thesis of the author [23] and extends the work presented in Dagnino et al. [26] in several ways: first, we consider a more natural and general notion of big-step semantics; we provide a detailed analysis of computations in big-step semantics; we define an additional construction based on coaxioms generalising the approach in Ancona et al. [9]; finally, we improve examples considering also imperative languages.

The rest of the paper is organised as follows. Section 2 recalls basic notions about inference systems and corules. Section 3 provides a definition of big-step semantics. Section 4 defines computations in big-step semantics as possibly infinite sequences of steps in a transition relation on partial evaluation trees. In this way we get a reference semantic model. Section 5 defines two constructions extending a given big-step semantics: one, based on traces, which explicitly models diverging computations and another which explicitly models stuck computations. Section 6 defines a third construction, modelling divergence just as a special result, by using appropriate corules. Section 7 shows how we can express two flavours of soundness against big-step semantics and provides proof techniques to show this property. Section 8 illustrates the proof technique on several examples. Finally, Section 9 concludes the paper, discussing related and future work.

2 PRELIMINARIES ON INFERENCE SYSTEMS AND CORULES

In this section, we recall standard notions about (co)inductive definitions by inference systems [3, 36, 51], which are used throughout the paper, and also their generalisation by corules, introduced by Ancona et al. [8], Dagnino [22, 23], which enable more flexible coinductive definitions. Corules will be only used in Sections 6 and 7 to properly model and reason about diverging computations in a big-step semantics.

Assume a set $\mathcal{U}$, named *universe*, whose elements are called *judgements*. An *inference system* $\mathcal{I}$ is a set of *(inference) rules*, which are pairs $\langle Pr, c\rangle$, where $Pr \subseteq \mathcal{U}$ the set of *premises* and $c \in \mathcal{U}$ the *conclusion* (a.k.a. *consequence*). As it is customary, rules are often written as fractions $\dfrac{Pr}{c}$. A rule with an empty set of premises is an *axiom*. A *proof tree* (a.k.a. *derivation*) for a judgement j in $\mathcal{I}$ is a tree whose nodes are (labeled with) judgements in $\mathcal{U}$, j is the root, and there is a node c with set of children Pr only if there is a rule $\langle Pr, c\rangle$ in $\mathcal{I}$. The *inductive* and the *coinductive interpretations* of $\mathcal{I}$, denoted $\mu[\![\mathcal{I}]\!]$ and $\nu[\![\mathcal{I}]\!]$, respectively, are the sets of judgements with, respectively a well-founded[1] and an arbitrary (well-founded or not) proof tree. We will write $\mathcal{I} \vdash_\mu j$ and $\mathcal{I} \vdash_\nu j$ when $j \in \mu[\![\mathcal{I}]\!]$ and $j \in \nu[\![\mathcal{I}]\!]$, respectively. Set-theoretically, we say that a subset $X \subseteq \mathcal{U}$ is *($\mathcal{I}$-)closed* if, for every rule $\langle Pr, j\rangle \in \mathcal{I}$, $Pr \subseteq X$ implies $j \in X$, and *($\mathcal{I}$-)consistent* if, for every $j \in X$, there is a rule $\langle Pr, j\rangle \in \mathcal{I}$ such that $Pr \subseteq X$. Then, it can be proved that $\mu[\![\mathcal{I}]\!]$ is the least closed subset and $\nu[\![\mathcal{I}]\!]$ is the largest consistent subset and this provides us with the following proof principles:

induction principle if $X \subseteq \mathcal{U}$ is closed then $\mu[\![\mathcal{I}]\!] \subseteq X$
coinduction principle if $X \subseteq \mathcal{U}$ is consistent then $X \subseteq \nu[\![\mathcal{I}]\!]$

We recall now the notion of inference system with corules [8, 22, 23], which mixes induction and coinduction in a specific way.

For a set $X \subseteq \mathcal{U}$, let $\mathcal{I}_{|X}$ denote the inference system obtained from $\mathcal{I}$ by keeping only rules with conclusion in X.

Definition 2.1 (Inference system with corules). An *inference system with corules*, or *generalised inference system*, is a pair $\langle \mathcal{I}, \mathcal{I}_{\mathrm{co}}\rangle$ where $\mathcal{I}$ and $\mathcal{I}_{\mathrm{co}}$ are inference systems, whose elements are called *rules* and *corules*, respectively. A corule with empty set of premises is a *coaxiom*. The *interpretation* $\nu[\![\mathcal{I}, \mathcal{I}_{\mathrm{co}}]\!]$ of such a pair is defined by $\nu[\![\mathcal{I}, \mathcal{I}_{\mathrm{co}}]\!] = \nu[\![\mathcal{I}_{|\mu[\![\mathcal{I}\cup\mathcal{I}_{\mathrm{co}}]\!]}]\!]$.

Thus, the interpretation $\nu[\![\mathcal{I}, \mathcal{I}_{\mathrm{co}}]\!]$ is basically *coinductive*, but restricted to a universe of judgements which is *inductively defined* by the (potentially) larger system $\mathcal{I} \cup \mathcal{I}_{\mathrm{co}}$. In proof-theoretic terms, $\nu[\![\mathcal{I}, \mathcal{I}_{\mathrm{co}}]\!]$ is the set of judgements which have an arbitrary (well-founded or not) proof tree in $\mathcal{I}$ whose nodes all have a well-founded proof tree in $\mathcal{I} \cup \mathcal{I}_{\mathrm{co}}$, that is, the (standard) inference system consisting of both rules and corules. We will write $\langle \mathcal{I}, \mathcal{I}_{\mathrm{co}}\rangle \vdash_\nu j$ when j is derivable in $\langle \mathcal{I}, \mathcal{I}_{\mathrm{co}}\rangle$, that is, $j \in \nu[\![\mathcal{I}, \mathcal{I}_{\mathrm{co}}]\!]$.

We illustrate these notions by a simple example. As usual, sets of rules are expressed by *meta-rules* with side conditions, and analogously sets of corules are expressed by *meta-corules* with side conditions. (Meta-)corules will be written with thicker lines, to be distinguished from (meta-)rules. The following inference system defines the maximal element of a list of natural numbers, where ε is the empty list, and $x{:}u$ the list with head x and tail u.

$$\frac{}{\mathsf{maxElem}(x{:}\varepsilon, x)} \qquad \frac{\mathsf{maxElem}(u, y)}{\mathsf{maxElem}(x{:}u, z)}\ z = \max(x, y)$$

The inductive interpretation is defined only on finite lists, since for infinite lists an infinite proof is needed. However, the coinductive interpretation allows the derivation of wrong judgements. For instance, let $L = 1 : 2 : 1 : 2 : 1 : 2 : \ldots$. Then, any judgement $\mathsf{maxElem}(L, x)$ with $x \geq 2$ can be derived, as illustrated by the following examples.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\mathsf{maxElem}(L,2)}}{\mathsf{maxElem}(2{:}L,2)}}{\mathsf{maxElem}(1{:}2{:}L,2)}}{\mathsf{maxElem}(2{:}L,2)}}{\mathsf{maxElem}(1{:}2{:}L,2)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\mathsf{maxElem}(L,5)}}{\mathsf{maxElem}(2{:}L,5)}}{\mathsf{maxElem}(1{:}2{:}L,5)}}{\mathsf{maxElem}(2{:}L,5)}}{\mathsf{maxElem}(1{:}2{:}L,5)}$$

[1] It is finite when sets of premises are finite.

By adding a corule (in this case a coaxiom), we add a constraint which forces the greatest element to belong to the list, so that wrong results are "filtered out":

$$\frac{}{\mathsf{maxElem}(x{:}\varepsilon, x)} \qquad \frac{\mathsf{maxElem}(u, y)}{\mathsf{maxElem}(x{:}u, z)}\ z = \max(x, y) \qquad \frac{}{\mathsf{maxElem}(x{:}u, x)}$$

Indeed, the judgement $\mathsf{maxElem}(1{:}2{:}L, 2)$ has the infinite proof tree shown above, and each node has a finite proof tree in the inference system extended by the corule:

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\mathsf{maxElem}(L, 2)}}{\mathsf{maxElem}(2{:}L, 2)}}{\mathsf{maxElem}(1{:}2{:}L, 2)} \qquad \dfrac{}{\mathsf{maxElem}(2{:}L, 2)}}{\mathsf{maxElem}(1{:}2{:}L, 2)}$$

On the other hand, the judgement $\mathsf{maxElem}(1{:}2{:}L, 5)$ has the infinite proof tree shown above, but has *no finite proof tree* in the inference system extended by the corule. Indeed, since 5 does not belong to the list, the corule can never be applied. Hence, this judgement cannot be derived in the inference system with corules. Finally, note that the judgement $\mathsf{maxElem}(1{:}2{:}L, 1)$ has a finite proof tree in the inference system extended by the corule, but has no proof tree in the system with no corules, as 1 is not an upper bound of the list. We refer to [8–10, 22, 23, 25] for other examples.

Let $\langle \mathcal{I}, \mathcal{I}_{\mathsf{co}} \rangle$ be a generalised inference system. The interpretation $\nu[\![\mathcal{I}, \mathcal{I}_{\mathsf{co}}]\!]$ can be characterised as the largest $\mathcal{I}$-consistent subset of $\mu[\![\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}]\!]$, and this provides us with the *bounded coinduction principle*, a generalisation of the standard coinduction principle.

Theorem 2.2 (Bounded coinduction). *Let $X \subseteq \mathcal{U}$. If X is $\mathcal{I}$-consistent and $X \subseteq \mu[\![\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}]\!]$, then $X \subseteq \nu[\![\mathcal{I}, \mathcal{I}_{\mathsf{co}}]\!]$.*

In other words, to prove that every judgement in X is derivable in $\langle \mathcal{I}, \mathcal{I}_{\mathsf{co}} \rangle$, we have to prove that every judgement in X has a well-founded proof tree in $\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}$ and every judgement in X is the conclusion of a rule whose premises are all in X.

3 DEFINING BIG-STEP SEMANTICS

As mentioned in the introduction, the corner stone of this paper is a formalisation of what a big-step semantics is, that captures its essential features, subsuming a large class of examples. This enables a general formal reasoning on an arbitrary big-step semantics.

Definition 3.1. A *big-step semantics* is a triple $\langle C, R, \mathcal{R} \rangle$ where:

- C is a set of *configurations* c.
- R is a set of *results* r. A *judgment* j is a pair written $c \Rightarrow r$, meaning that configuration c evaluates to result r. Set $C(j) = c$ and $R(j) = r$.
- $\mathcal{R}$ is a set of *(big-step) rules* ρ of shape
$$\frac{j_1 \quad \cdots \quad j_n}{c \Rightarrow r} \qquad \text{also written in } \textit{inline format}\text{: } \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$$
where $j_1 \dots j_n$, with $n \geq 0$, is a sequence of premises. Set $C(\rho) = c$, $R(\rho) = r$ and, for $i \in 1..n$, $C(\rho, i) = C(j_i)$ and $R(\rho, i) = R(j_i)$.
We require $\mathcal{R}$ to satisfy the *bounded premises* condition:
BP for every $c \in C$, there exists $b_c \in \mathbb{N}$ such that, for each $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$, $n \leq b_c$.

$$\begin{array}{lcll} e & ::= & x \mid v \mid e_1\ e_2 \mid \mathsf{succ}\ e \mid e_1 \oplus e_2 & \text{expression} \\ v & ::= & n \mid \lambda x.e & \text{value} \end{array}$$

$$(\textsc{val})\ \frac{}{v \Rightarrow v} \qquad (\textsc{app})\ \frac{e_1 \Rightarrow \lambda x.e \quad e_2 \Rightarrow v_2 \quad e[v_2/x] \Rightarrow v}{e_1\ e_2 \Rightarrow v}$$

$$(\textsc{succ})\ \frac{e \Rightarrow n}{\mathsf{succ}\ e \Rightarrow n+1} \qquad (\textsc{choice})\ \frac{e_i \Rightarrow v}{e_1 \oplus e_2 \Rightarrow v}\ \ i = 1,2$$

(VAL) $\mathsf{rule}(\varepsilon,\ v,\ v)$
(APP) $\mathsf{rule}(e_1 \Rightarrow \lambda x.e\ \ e_2 \Rightarrow v_2\ \ e[v_2/x] \Rightarrow v,\ e_1\ e_2,\ v)$
(SUCC) $\mathsf{rule}(e \Rightarrow n,\ \mathsf{succ}\ e,\ n+1)$
(CHOICE) $\mathsf{rule}(e_i \Rightarrow v,\ e_1 \oplus e_2,\ v)\ i = 1,2$

Fig. 1. Example of big-step semantics

We will use the inline format, more concise and manageable, for the development of the meta-theory, e.g., in constructions.

Big-step rules, as defined above, are very much like inference rules (cf. Section 2), but they carry slightly more structure with respect to them. Notably, premises are a sequence rather than a set, that is, they are ordered and there can be repeated premises. Such additional structure, however, does not affect derivability, namely, the inference operator and so the interpretations of such rules. Therefore, given a big-step semantics $\langle C, R, \mathcal{R}\rangle$, slightly abusing the notation, we denote by $\mathcal{R}$ the inference system obtained by forgetting such additional structure, and define, as usual, the *semantic relation* as the inductive interpretation of $\mathcal{R}$. Then, we write $\mathcal{R} \vdash_\mu c \Rightarrow r$ when the judgment $c \Rightarrow r$ is derivable in $\mathcal{R}$.

Even though the additional structure of big-step rules does not affect the semantic relation they define, it is crucial to develop the meta-theory, allowing abstract reasoning about an arbitrary big-step semantics. It will be used in all results in this paper: to define computations in big-step semantics, then to provide constructions yelding extended semantics able to distinguish stuck and diverging computations and, finally, to define proof techniques for soundness. Indeed, as premises are a sequence, we know in which order configurations in the premises should be evaluated.

In practice, the (infinite) set of rules $\mathcal{R}$ is described by a finite set of meta-rules, each one with a finite number of premises. As a consequence, for each configuration, the number of premises of rules with such a configuration in the conclusion is not only finite but *bounded*. Since we have no notion of meta-rule, we explicitly require this feature (relevant in the following) by the bounded premises (BP) condition.

We end this section by illustrating the above definitions and conditions on a simple example: a λ-calculus with constants for natural numbers, successor and non-deterministic choice, shown in Fig. 1. We denote by x variables and by n natural number constants. It is immediate to see this example as an instance of Definition 3.1:

- Configurations and results are expressions, and values, respectively.[2]
- To have the set of (meta-)rules in our required shape, abbreviated in inline format in the bottom section of the figure, we have only to assume an order on premises of rule (APP).

Remark 3.2. The order of premises chosen for rule (APP) in Fig. 1 formalises the evaluation strategy for an application $e_1\ e_2$ where first (1) evaluates e_1, then (2) checks that the value of e_1 is a λ-abstraction, finally (3) evaluates e_2. That is, left-to-right evaluation with early error detection. Other strategies can be obtained by choosing a different order or by adjusting big-step rules. Notably, right-to-left evaluation (3)-(1)-(2) can be expressed by just swapping the first two

[2]In general, configurations may include additional components and results are not necessarily particular configurations, see, *e.g.*, Section 8.2.

premises, that is:

$$(\textsc{app-r})\ \mathsf{rule}(e_2 \Rightarrow v_2 \quad e_1 \Rightarrow \lambda x.e \quad e[v_2/x] \Rightarrow v,\ e_1\ e_2,\ v)$$

Left-to-right evaluation with late error detection (1)-(3)-(2) can be expressed as follows:

$$(\textsc{app-late})\ \mathsf{rule}(e_1 \Rightarrow v_1 \quad e_2 \Rightarrow v_2 \quad v_1 \Rightarrow \lambda x.e \quad e[v_2/x] \Rightarrow v,\ e_1\ e_2,\ v)$$

We can even opt for a non-deterministic approach by taking more than one rule among (app), (app-r) and (app-late). As said above, these different choices do not affect the semantic relation inductively defined by the inference system, which is always the same. However, they will affect computations and thus the extended semantics distinguishing stuck computation and non-termination. Indeed, if the evaluation of e_1 and e_2 is stuck and non-terminating, respectively, we should obtain a stuck computation with rule (app) and non-termination with rule (app-r); further, if e_1 evaluates to a natural constant and e_2 diverges, we should obtain a stuck computation with rule (app) and non-termination with rule (app-late).

In summary, to see a typical big-step semantics as an instance of our definition, it is enough to identify configurations and results and to assume an order (or more than one) on premises.

4 COMPUTATIONS IN BIG-STEP SEMANTICS

Intuitively, the evaluation of a configuration c is a *dynamic* process and, as such, it may either successfully terminate producing the final result, or get stuck, or never terminate. However, a big-step semantics just tells us whether a configuration c evaluates to a certain result r, without describing the dynamics of such evaluation process. This is nice, because it allows us to abstract away details about intermediate states in the evaluation process, but it makes quite difficult to reason about concepts like non-termination and stuckness, since they refer to computations and we do not even know what a computation is in a big-step semantics.

In this section, we show that, given a big-step semantics as defined in Definition 3.1, we can recover the dynamics of the evaluation, by defining *computations*, which, in a sense, are implicit in a big-step specification. To this end, we extend the big-step semantics, so that we can represent partial (or incomplete) evaluations, modelling intermediate states of the evaluation process. Then, we model the dynamics by a transition relation beween such partial evaluations, hence, as usual, a computation will be a (possibly infinite) sequence of transitions.

Let us assume a big-step semantics $\langle C, R, \mathcal{R}\rangle$. As said above, the first step is to extend such semantics to model partial evaluations. To this end, first of all, we introduce a special result ?, so that a judgment $c \Rightarrow ?$ (called *incomplete*, whereas a judgment $c \Rightarrow r$ is *complete*) means that the evaluation of c is not completed yet. Set $R_? = R + \{?\}$ whose elements are ranged over by $r_?$. We now define an augmented set of rules $\mathcal{R}_?$ to properly handle the new result ?:

Definition 4.1 (Rules for partial evaluation). The set of rules $\mathcal{R}_?$ is obtained from $\mathcal{R}$ by adding the following rules:

start rules For each configuration $c \in C$, define rule $\mathsf{ax}_?(c)$ as $\dfrac{}{c \Rightarrow ?}$.

partial rules For each rule $\rho = \mathsf{rule}(j_1 \ldots j_n,\ c,\ r)$ in $\mathcal{R}$, index $i \in 1..n$, and $r_? \in R_?$, define rule $\mathsf{pev}_?(\rho, i, r_?)$ as

$$\frac{j_1 \quad \ldots \quad j_{i-1} \quad C(j_i) \Rightarrow r_?}{c \Rightarrow ?}$$

Intuitively, start rules allow us to begin the evaluation of any configuration, while partial rules allow us to partially apply a rule from $\mathcal{R}$ to derive a partial judgement. Note that the last premise of a partial rule can be either complete ($r_? \in R$) or incomplete ($r_? = ?$), in the latter case we also call it a *?-propagation* rule, since it propagates ? from premises to the conclusion.

$$\frac{}{e \Rightarrow ?} \qquad \frac{e \Rightarrow v_?}{\mathsf{succ}\, e \Rightarrow ?} \qquad \frac{e_i \Rightarrow v_?}{e_1 \oplus e_2 \Rightarrow ?}\ i = 1,2$$

$$\frac{e_1 \Rightarrow v_?}{e_1\ e_2 \Rightarrow ?} \qquad \frac{e_1 \Rightarrow \lambda x.e \quad e_2 \Rightarrow v_?}{e_1\ e_2 \Rightarrow ?} \qquad \frac{e_1 \Rightarrow \lambda x.e \quad e_2 \Rightarrow v_2 \quad e[v_2/x] \Rightarrow v_?}{e_1\ e_2 \Rightarrow ?}$$

Fig. 2. Rules for ? for the λ-calculus in Fig. 1.

It is important to observe that the construction described above yields a triple $\langle C, R_?, \mathcal{R}_? \rangle$, which is a big-step semantics according to Definition 3.1. [3] In Fig. 2 we report rules added by the construction in Definition 4.1 to the big-step semantics of the λ-calculus in Fig. 1.

Given a big-step semantics $\langle C, R, \mathcal{R} \rangle$, using rules in $\mathcal{R}$, we can build trees called *evaluation trees*. Such trees are very much like proof trees for an inference system, with the only difference that evaluation trees are *ordered* trees, because premises of big-step rules are a sequence. Roughly, an evaluation tree is an ordered tree with nodes labelled by semantic judgements, such that for each node labelled by $c \Rightarrow r$ with sequence of children $j_1, \ldots, j_n$, there is a rule $\mathsf{rule}(j_1 \ldots j_n,\ c,\ r)$ in $\mathcal{R}$.

An evaluation tree for $\langle C, R_?, \mathcal{R}_? \rangle$ is called a *partial evaluation tree*, as it can contain incomplete judgements. We say that a partial evaluation tree is *complete* if it only contains complete judgments, it is *incomplete* otherwise. Finite partial evaluation trees indeed model possibly incomplete evaluation of configurations, namely, the intermediate states of the evaluation process, because big-step rules can be partially applied. Hence, they are the fundamental building block, which will allow us to define computations in big-step semantics.

In the next subsection we will give a formal definition of (partial) evaluation trees, similar to that of proof trees [22–24]. This formal definition is needed to state some results and to carry out proofs in a rigorous way, and it is not essential to follow the rest of the paper, hence the reader not interested in formal details can skip it, relying on the above semiformal definition.

4.1 The structure of partial evaluation trees

We give a formal account of (partial) evaluation trees, which is useful to state and prove technical results in the next sections. This development is based on the definition and properties of trees provided by Courcelle [20], adjusted to our specific setting.

Let $\mathbb{N}_{>0}$ be the set of positive natural numbers, $\mathbb{N}_{>0}^\star$ the set of finite sequences of positive natural numbers and $\mathcal{L}$ a set of labels. An *ordered tree* labelled in $\mathcal{L}$ is a partial function $\tau : \mathbb{N}_{>0}^\star \to \mathcal{L}$ such that $\mathsf{dom}(\tau)$ is not empty, and, for each $\alpha \in \mathbb{N}_{>0}^\star$ and $n \in \mathbb{N}_{>0}$, if $\alpha n \in \mathsf{dom}(\tau)$ then $\alpha \in \mathsf{dom}(\tau)$ and, for all $k \leq n$, $\alpha k \in \mathsf{dom}(\tau)$. Given an ordered tree τ and $\alpha \in \mathsf{dom}(\tau)$, set $\mathsf{br}_\tau(\alpha) = \sup\{n \in \mathbb{N} \mid \alpha n \in \mathsf{dom}(\tau)\}$ the *branching* of τ at α, and $\tau_{|_\alpha}$ the *subtree* of τ rooted at α, that is, $\tau_{|_\alpha}(\beta) = \tau(\alpha\beta)$, for all $\beta \in \mathbb{N}_{>0}^\star$. The *root* of τ is $\mathsf{r}(\tau) = \tau(\varepsilon)$ and obviously we have $\tau = \tau_{|_\varepsilon}$. Finally, we write $\dfrac{\tau_1 \quad \ldots \quad \tau_n}{x}$ for the tree τ defined by $\tau(\varepsilon) = x$, and $\tau(i\alpha) = \tau_i(\alpha)$ for all $i \in 1..n$. Since in the following we will only deal with ordered trees, we will refer to them just as trees.

Let us assume a big-step semantics $\langle C, R, \mathcal{R} \rangle$. Assume also that labels in $\mathcal{L}$ are semantic judgments $c \Rightarrow r$, then we can define evaluation trees as follows:

Definition 4.2. A tree $\tau : \mathbb{N}_{>0}^\star \to \mathcal{L}$ is an *evaluation tree* in $\langle C, R, \mathcal{R} \rangle$, if, for each $\alpha \in \mathsf{dom}(\tau)$ with $\tau(\alpha) = c \Rightarrow r$, there is $\mathsf{rule}(\tau(\alpha 1) \ldots \tau(\alpha \mathsf{br}_\tau(\alpha)),\ c,\ r) \in \mathcal{R}$.

[3] The condition (BP) is satisfied as the number of premises of the additional rules is bounded by that of a rule in the original semantics.

Note that, starting from an evaluation tree τ, we can construct a proof tree for the inference system denoted by $\mathcal{R}$, by forgetting the order on sibling nodes and removing duplicated children. Therefore, if τ is a finite evaluation tree with $\mathsf{r}(\tau) = c \Rightarrow r$, then $\mathcal{R} \vdash_\mu c \Rightarrow r$ holds.

Definition 4.3. A *partial evaluation tree* in $\langle C, R, \mathcal{R}\rangle$ is an evaluation tree in $\langle C, R_?, \mathcal{R}_?\rangle$.

The following proposition assures two key properties of partial evaluation trees. First, if there is some ?, then it is propagated to ancestor nodes. Second, for each level of the tree there is at most one ?. We set $|\alpha|$ the length of $\alpha \in \mathbb{N}^\star_{>0}$.

Proposition 4.4. *Let τ be a partial evaluation tree, then the following hold:*

(1) for all $\alpha n \in dom(\tau)$, if $R_?(\tau(\alpha n)) = ?$ then $R_?(\tau(\alpha)) = ?$.
(2) for all $n \in \mathbb{N}$, there is at most one $\alpha \in dom(\tau)$ with $|\alpha| = n$ such that $R_?(\tau(\alpha)) = ?$.

Proof. To prove Item 1, we just have to note that the only rules having a premise j with $R_?(j) = ?$ are ?-propagation rules, which also have conclusion j' with $R_?(j') = ?$; hence the thesis is immediate. To prove Item 2, we proceed by induction on n. For $n = 0$, there is only one $\alpha \in \mathbb{N}^\star_{>0}$ with $|\alpha| = 0$ (the empty sequence), hence the thesis is trivial. Consider $\alpha = \alpha' k \in \mathsf{dom}(\tau)$ with $|\alpha| = n + 1$. If $R_?(\tau(\alpha)) = ?$, then, by Item 1, $R_?(\tau(\alpha')) = ?$, and, by induction hypothesis, α' is the only sequence of length n in $\mathsf{dom}(\tau)$ with this property. Therefore, another node $\beta \in \mathsf{dom}(\tau)$, with $|\beta| = n + 1$ and $R_?(\tau(\beta)) = ?$, must satisfy $\beta = \alpha' h$ for some $h \in \mathbb{N}_{>0}$; hence, since τ is a partial evaluation tree, $\tau(\alpha)$ and $\tau(\beta)$ are two premises of the same rule with ? as result, thus they must coincide, since all rules in $\mathcal{R}_?$ have at most one premise with ?. □

Corollary 4.5. *Let τ be a partial evaluation tree, then $R_?(\mathsf{r}(\tau)) \in R$ if and only if τ is complete.*

We can define a relation[4], denoted by $\sqsubseteq$, on trees labelled by possibly incomplete judgements, as follows:

Definition 4.6. Let τ and τ' be trees labelled by possibly incomplete semantic judgements. Define $\tau \sqsubseteq \tau'$ if and only if $\mathsf{dom}(\tau) \subseteq \mathsf{dom}(\tau')$ and, for all $\alpha \in \mathsf{dom}(\tau)$, $C(\tau(\alpha)) = C(\tau'(\alpha))$ and $R_?(\tau(\alpha)) \in R$ implies ${\tau}_{|_\alpha} = {\tau'}_{|_\alpha}$.

Intuitively, $\tau \sqsubseteq \tau'$ means that τ' can be obtained from τ by adding new branches or replacing some ?s with results. We use $\sqsubset$ for the strict version of $\sqsubseteq$. Note that, if $\tau \sqsubseteq \tau'$, then, for all $\alpha \in \mathbb{N}^\star_{>0}$, $\tau'(\alpha)$ is *more defined* than $\tau(\alpha)$, because, either $\tau(\alpha)$ is undefined, or $\tau(\alpha)$ is incomplete and $C(\tau(\alpha)) = C(\tau'(\alpha))$, or $\tau(\alpha) = \tau'(\alpha)$.

It is easy to check that $\sqsubseteq$ is a partial order and, if $\tau \sqsubseteq \tau'$, then, for all $\alpha \in \mathsf{dom}(\tau)$, ${\tau}_{|_\alpha} \sqsubseteq {\tau'}_{|_\alpha}$. The following proposition shows some, less trivial, properties of $\sqsubseteq$.

Proposition 4.7. *The following properties hold:*

(1) for all trees τ and τ', if $\tau \sqsubseteq \tau'$ and $R_?(\mathsf{r}(\tau)) \in R$, then $\tau = \tau'$
(2) for each increasing sequence $(\tau_i)_{i\in\mathbb{N}}$ of trees, there is a least upper bound $\tau = \bigsqcup \tau_n$.

Proof. Item 1 is immediate by definition of $\sqsubseteq$. To prove Item 2, first note that, since for all $n \in \mathbb{N}$, $\tau_n \sqsubseteq \tau_{n+1}$, for all $\alpha \in \mathbb{N}^\star_{>0}$ we have that, for all $n \in \mathbb{N}$, if $\tau_n(\alpha)$ is defined, then, for all $k \geq n$, $C(\tau_k(\alpha)) = C(\tau_n(\alpha))$, and, if $R_?(\tau_n(\alpha)) \in R$, then $\tau_k(\alpha) = \tau_n(\alpha)$. Hence, for all $n \in \mathbb{N}$, there are only three possibilities for $\tau_n(\alpha)$: it is either undefined, or equal to $c \Rightarrow ?$, or equal to $c \Rightarrow r$, where c and r are always the same. Let us denote by k_α the least index n where $\tau_n(\alpha)$ is most defined, that is, if $\tau_n(\alpha)$ is always undefined, then $k_\alpha = 0$, if $\tau_n(\alpha)$ is eventually always equal to $c \Rightarrow ?$, then k_α

[4]This is a slight variation of similar relations on trees considered by Courcelle [20], Dagnino [22].

is the least n where $\tau_n(\alpha)$ is defined, and, if $\tau_n(\alpha)$ is eventually always equal to $c \Rightarrow r$, then k_α is the least n where $\tau_n(\alpha) = c \Rightarrow r$. Therefore, for all $n \geq k_\alpha$, we have that $\tau_n(\alpha) = \tau_{k_\alpha}(\alpha)$.

Consider a tree τ defined by $\tau(\alpha) = \tau_{k_\alpha}(\alpha)$. It is easy to check that $\mathrm{dom}(\tau) = \bigcup_{n\in\mathbb{N}} \mathrm{dom}(\tau_n)$. We now check that, for all $n \in \mathbb{N}$, $\tau_n \sqsubseteq \tau$. For all $\alpha \in \mathrm{dom}(\tau_n)$, we have $\alpha \in \mathrm{dom}(\tau)$ and we distinguish two cases:

- if $\tau_n(\alpha) = c \Rightarrow ?$, then, since either $\tau_n \sqsubseteq \tau_{k_\alpha}$ or $\tau_{k_\alpha} \sqsubseteq \tau_n$ and $\alpha \in \mathrm{dom}(\tau_{k_\alpha})$, we get $C(\tau(\alpha)) = C(\tau_{k_\alpha}(\alpha)) = C(\tau_n(\alpha)) = c$;
- if $\tau_n(\alpha) = c \Rightarrow r$, then $k_\alpha \leq n$, hence, since $\tau_{k_\alpha} \sqsubseteq \tau_n$, we get $C(\tau(\alpha)) = C(\tau_{k_\alpha}(\alpha)) = C(\tau_n(\alpha)) = c$, thus we have only to check that $\tau_{n\,|\alpha} = \tau_{|\alpha}$. To prove this point, consider $\beta \in \mathrm{dom}(\tau_{|\alpha})$, then, by Corollary 4.5, we have $\tau_{|\alpha}(\beta) = \tau(\alpha\beta) = c' \Rightarrow r'$, hence, since for all $h \geq k_\alpha$ we have $\tau_{k_\alpha\,|\alpha} = \tau_{h\,|\alpha}$, we get $\alpha\beta \in \mathrm{dom}(\tau_h)$ and $\tau_{h\,|\alpha}$ is complete, thus $k_{\alpha\beta} \leq k_\alpha$. Therefore, $\tau_{k_{\alpha\beta}} \sqsubseteq \tau_{k_\alpha} \sqsubseteq \tau_n$ and so we get $\tau_{k_{\alpha\beta}\,|\alpha\beta} = \tau_{n\,|\alpha\beta}$, which implies that $\tau_{n\,|\alpha}(\beta) = \tau_{k_{\alpha\beta}}(\alpha\beta) = \tau(\alpha\beta)$, as needed.

This proves that τ is an upper bound of the sequence, we have still to prove that it is the least one. To this end, let τ' be an upper bound of the sequence: we have to show that $\tau \sqsubseteq \tau'$. Since τ' is an upper bound, for all $n \in \mathbb{N}$ we have $\mathrm{dom}(\tau_n) \subseteq \mathrm{dom}(\tau')$, hence $\mathrm{dom}(\tau) \subseteq \mathrm{dom}(\tau')$, and, especially, for all $\alpha \in \mathbb{N}^\star_{>0}$ we have $\tau_{k_\alpha} \sqsubseteq \tau'$. Hence, for all $\alpha \in \mathrm{dom}(\tau)$, we have $C(\tau(\alpha)) = C(\tau_{k_\alpha}(\alpha)) = C(\tau'(\alpha))$, and, if $R_?(\tau(\alpha)) = r$, since $\tau_{k_\alpha} \sqsubseteq \tau$ and $\tau_{k_\alpha} \sqsubseteq \tau'$, we have $\tau_{k_\alpha\,|\alpha} = \tau_{|\alpha}$ and $\tau_{k_\alpha\,|\alpha} = \tau'_{|\alpha}$, hence $\tau_{|\alpha} = \tau'_{|\alpha}$, as needed. □

Obviously, this relation restricts to partial evaluation trees and, more importantly, the set of partial evaluation trees is closed with respect to least upper bound for $\sqsubseteq$, as the next proposition shows.

Proposition 4.8. *For each increasing sequence $(\tau_n)_{n\in\mathbb{N}}$ of partial evaluation trees, the least upper bound $\bigsqcup \tau_n$ is a partial evaluation tree as well.*

Proof. Set $\tau = \bigsqcup \tau_n$. We have to show that for every node $\alpha \in \mathrm{dom}(\tau)$ there is a rule in $\mathcal{R}_?$ with conclusion $\tau(\alpha)$ and premises the (labels of) the children of α in τ.

Recall from Proposition 4.7 (2) that $\tau(\alpha) = \tau_{k_\alpha}(\alpha)$, where $k_\alpha \in \mathbb{N}$ is the least index n where $\tau_n(\alpha)$ is most defined. Note that, for all $\alpha \in \mathrm{dom}(\tau)$, $\mathrm{br}_\tau(\alpha)$ is finite. Indeed, by definition of τ and since the sequence is increasing, we have $\mathrm{br}_\tau(\alpha) = \sup\{\mathrm{br}_{\tau_n}(\alpha) \mid n \geq k_\alpha\}$, and $\mathrm{br}_{\tau_n}(\alpha)$ is the number of premises of a rule, for all $n \geq k_\alpha$; all such rules have the same configuration in the conclusion $C(\tau_n(\alpha)) = C(\tau(\alpha))$, hence, by condition (BP) in Definition 3.1, there is $b \in \mathbb{N}$ such that $\mathrm{br}_{\tau_n}(\alpha) \leq b$, thus $\mathrm{br}_\tau(\alpha) \leq b$. Then, the set $K = \{k_\alpha\} \cup \{k_{\alpha i} \mid i \in 1..\mathrm{br}_\tau(\alpha)\}$ is finite and $n = \max K$ is finite, hence, as $n \geq k_\alpha$ and $n \geq k_{\alpha i}$, for all $i \in 1..\mathrm{br}_\tau(\alpha)$, we have $\tau_n(\alpha) = \tau(\alpha)$ and $\tau_n(\alpha i) = \tau(\alpha i)$, for all $i \in 1..\mathrm{br}_\tau(\alpha)$. Therefore, $\langle \tau(\alpha 1) \dots \tau(\alpha \mathrm{br}_\tau(\alpha)), \tau(\alpha)\rangle = \langle \tau_n(\alpha 1) \dots \tau_n(\alpha \mathrm{br}_\tau(\alpha)), \tau_n(\alpha)\rangle \in \mathcal{R}_?$, since τ_n is a partial evaluation tree. Thus, by Definition 4.3, τ is a partial evaluation tree. □

As already mentioned, finite partial evaluation trees model possibly incomplete evaluations. Then, the relation $\sqsubseteq$ models refinement of the evaluation, because if $\tau \sqsubseteq \tau'$, where τ and τ' are finite partial evaluation trees, τ' is "more detailed" than τ. In a sense, $\sqsubseteq$ on finite partial evaluation trees abstracts the process of evaluation itself, as we will make precise in the next section.

What about infinite trees? Similarly to what we have discussed in the introduction, there are many infinite partial evaluation trees which are difficult to interpret. For instance, using rules in Fig. 1 and Fig. 2, we can construct the

following infinite tree for all $v_?$, where $\Omega = (\lambda x.x\,x)\,(\lambda x.x\,x)$:

$$\dfrac{\dfrac{}{\lambda x.x\,x \Rightarrow \lambda x.x\,x} \quad \dfrac{}{\lambda x.x\,x \Rightarrow \lambda x.x\,x} \quad \dfrac{\vdots}{\Omega = (x\,x)[\lambda x.x\,x/x] \Rightarrow v_?}}{\Omega \Rightarrow v_?}$$

Among all such trees there are some "good" ones, we call them *well-formed*. Well-formed infinite partial evaluation trees arise as limits of strictly increasing sequences of finite partial evaluation trees, hence, in a sense, they model the limit of the evaluation process. namely, non-termination.

Definition 4.9. An infinite partial evaluation tree τ is *well-formed* if, for all $\alpha \in \mathsf{dom}(\tau)$, if $R_?(\tau(\alpha)) \in R$, then $\tau_{|_\alpha}$ is finite.

In other words, in a well-formed partial evaluation tree all complete subtrees are finite. The next proposition, together with Proposition 4.4, implies that a well-formed tree contains a unique infinite path, which is entirely labelled by incomplete judgments. A similar property on infinite derivations will be enforced by corules in the semantics for divergence in Section 6.

Proposition 4.10. *If τ is a well-formed infinite partial evaluation tree then, for all $n \in \mathbb{N}$, there is $\alpha \in dom(\tau)$ such that $|\alpha| = n$ and $R_?(\tau(\alpha)) = ?$.*

Proof. The proof is by induction on n. For $n = 0$, we have $R_?(\mathsf{r}(\tau)) = ?$, since, otherwise, we would have $R_?(\mathsf{r}(\tau)) = r$, hence, by Definition 4.9, $\tau = \tau_{|_\varepsilon}$ would be finite, while τ is infinite by hypothesis.

For $n = k + 1$, by induction hypothesis, we know there is $\alpha \in \mathsf{dom}(\tau)$ such that $|\alpha| = k$ and $R_?(\tau(\alpha)) = ?$. For all $\beta \in \mathsf{dom}(\tau)$ with $\beta = \alpha' h$, $|\alpha'| = k$, $\alpha' \neq \alpha$, we have $R_?(\tau(\beta)) \in R$, because, if $R_?(\tau(\beta)) = ?$, then also $R_?(\tau(\alpha')) = ?$, by Proposition 4.4, and, again by Proposition 4.4, this implies $\alpha' = \alpha$, which is absurd. As a consequence, for all such β, we have that $\tau_{|_\beta}$ is finite, as τ is well-formed.

Then, we focus on children of α, splitting cases over $\mathsf{br}_\tau(\alpha)$. If $\mathsf{br}_\tau(\alpha) = 0$, then α has no children and so τ is finite, which is absurd. If $h = \mathsf{br}_\tau(\alpha) > 0$, then, if $R_?(\tau(\alpha h)) \in R$, since τ is a partial evaluation tree, we get $R_?(\tau(\alpha h')) \in R$ for all $h' \leq h$, hence τ is again finite, which is absurd. Therefore, $R_?(\tau(\alpha h)) = ?$, as needed. □

The following result shows that well-formed partial evaluation trees are exactly the least upper bounds of strictly increasing sequences of finite partial evaluation trees.

Proposition 4.11. *The following properties hold:*

(1) for each strictly increasing sequence $(\tau_n)_{n\in\mathbb{N}}$ of finite partial evaluation trees, the least upper bound $\bigsqcup \tau_n$ is infinite and well-formed;

(2) for each well-formed infinite partial evaluation tree τ, there is a strictly increasing sequence $(\tau_n)_{n\in\mathbb{N}}$ of finite partial evaluation trees such that $\tau = \bigsqcup \tau_n$.

Proof. To prove Item 1, set $\tau = \bigsqcup \tau_n$, then, by Proposition 4.8, we have that τ is a partial evaluation tree, hence we have only to check that it is infinite and well-formed.

Note that τ is infinite if and only if $\mathsf{dom}(\tau) = \bigcup_{n\in\mathbb{N}} \mathsf{dom}(\tau_n)$ is infinite. To prove this, it suffices to observe that, for all $n \in \mathbb{N}$, there is $h > n$ such that $\mathsf{dom}(\tau_n) \subset \mathsf{dom}(\tau_h)$, namely, there is $\alpha \in \mathsf{dom}(\tau_h)$ such that $\alpha \notin \mathsf{dom}(\tau_n)$. This can be proved by induction on the number of ? in τ_n, denoted by $N_?(\tau_n)$, which is finite as τ_n is finite. This follows

because, if $\mathsf{dom}(\tau_n) = \mathsf{dom}(\tau_{n+1})$, we have $N_?(\tau_{n+1}) < N_?(\tau_n)$, since $\tau_n \sqsubset \tau_{n+1}$ implies that there is at least one node $\alpha \in \mathsf{dom}(\tau_n)$ such that $R_?(\tau_n(\alpha)) = ?$ and $R_?(\tau_{n+1}(\alpha)) = r$, thus we can apply the induction hypothesis.

To show that τ is well-formed, first recall that, for all $\alpha \in \mathsf{dom}(\tau)$, we have $\tau(\alpha) = \tau_n(\alpha)$ for some $n \in \mathbb{N}$. Then, for all $\alpha \in \mathsf{dom}(\tau)$ such that $\tau(\alpha) = c \Rightarrow r$, since $\tau_n \sqsubseteq \tau$ and $\tau_n(\alpha) = \tau(\alpha)$, by definition of $\sqsubseteq$, we get ${\tau_n}_{|_\alpha} = \tau_{|_\alpha}$; hence, $\tau_{|_\alpha}$ is finite and so τ is well-formed.

To prove Item 2, for all $n \in \mathbb{N}$, consider the partial evaluation tree τ_n obtained by "cutting" τ at level n and defined as follows. Let $\alpha_n \in \mathsf{dom}(\tau)$ be the node such that $|\alpha_n| = n$ and $R_?(\tau(\alpha_n)) = ?$ (which exists by Proposition 4.10 as τ is well-formed and it is unique thanks to Proposition 4.4 (2)), then define $\tau_n(\beta) = \tau(\beta)$ for all $\beta \neq \alpha_n\beta'$, with $\beta' \in \mathbb{N}^+_{>0}$, and undefined otherwise. We have $\tau_n \sqsubseteq \tau_{n+1}$, since, by Proposition 4.4 (1), $\alpha_{n+1} = \alpha_n i$ for some $i \in \mathbb{N}_{>0}$. Finally, by construction, we have $\tau = \bigsqcup \tau_n$, as needed. □

This important result will be used in the next sections to prove correctness of extended big-step semantics explicitly modelling divergence.

4.2 The transition relation

As already mentioned, finite partial evaluation trees nicely model intermediate states in the evaluation process of a configuration. We now make this precise by defining a transition relation $\longrightarrow_{\mathcal{R}}$ between them, such that, starting from the initial partial evaluation tree $\dfrac{}{c \Rightarrow ?}$, we derive a sequence where, intuitively, at each step we detail the evaluation. In this way, a sequence ending with a complete tree (a tree containing no ?) models succesfully terminating computation, whereas an infinite sequence (tending to an infinite partial evaluation tree) models divergence, and a sequence reaching an incomplete tree which cannot further move models a stuck computation.

The one-step transition relation $\longrightarrow_{\mathcal{R}}$ is inductively defined by the rules in Fig. 3. To make the definition clearer, we explicitly annotate the tree with the rule in $\mathcal{R}_?$ applied to derive the root of the tree from its children. In the figure, $\#\rho$ denotes the number of premises of ρ. Finally, $\sim_i$ is the *equality up-to an index* of rules, defined below:

Definition 4.12. Let $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ and $\rho' = \mathsf{rule}(j'_1 \dots j'_m,\ c',\ r')$ be rules in $\mathcal{R}$. Then, for any index $i \in 1..\min(n, m)$, define $\rho \sim_i \rho'$ if and only if

- $c = c'$,
- for all $k < i$, $j_k = j'_k$, and
- $C(j_i) = C(j'_i)$.

In other words, this equivalence models the fact that rules ρ and ρ' represent the same computation until the i-th configuration included.

Intuitively, each transition step makes "less incomplete" the partial evaluation tree. Notably, transition rules apply only to nodes labelled by incomplete judgements ($c \Rightarrow ?$), whereas subtrees whose root is a complete judgement ($c \Rightarrow r$) cannot move. In detail:

- If the last applied rule is $\mathsf{ax}_?(c)$, we have to find a rule ρ with c in the conclusion and, if it has no premises we just return $R(\rho)$ as result, otherwise we start evaluating the first premise of such rule.
- If the last applied rule is $\mathsf{pev}_?(\rho, i, r)$, then all subtrees are complete, hence, to continue the evaluation, we have to find another rule ρ', having, for each $k \in 1..i$, as k-th premise the root of τ_k. Then there are two possibilities: if there is an $i+1$-th premise, we start evaluating it, otherwise, we return $R(\rho')$ as result.

$$\text{(TR-1)}\quad \text{(ax}_?(c))\ \frac{}{c \Rightarrow ?} \longrightarrow_{\mathcal{R}} \text{(}\rho\text{)}\ \frac{}{c \Rightarrow r} \quad \begin{array}{l}\#\rho = 0\\ C(\rho) = c\\ R(\rho) = r\end{array}$$

$$\text{(TR-2)}\quad \text{(ax}_?(c))\ \frac{}{c \Rightarrow ?} \longrightarrow_{\mathcal{R}} \text{(pev}_?(\rho,1,?))\ \frac{c' \Rightarrow ?}{c \Rightarrow ?} \quad \begin{array}{l}\#\rho > 0\\ C(\rho) = c\\ C(\rho,1) = c'\end{array}$$

$$\text{(TR-3)}\quad \text{(pev}_?(\rho,i,r))\ \frac{\tau_1\ \ldots\ \tau_i}{c \Rightarrow ?} \longrightarrow_{\mathcal{R}} \text{(}\rho'\text{)}\ \frac{\tau_1\ \ldots\ \tau_i}{c \Rightarrow r'} \quad \begin{array}{l}\rho' \sim_i \rho\\ R(\rho', i) = r\\ \#\rho' = i\\ R(\rho') = r'\end{array}$$

$$\text{(TR-4)}\quad \text{(pev}_?(\rho,i,r))\ \frac{\tau_1\ \ldots\ \tau_i}{c \Rightarrow ?} \longrightarrow_{\mathcal{R}} \text{(pev}_?(\rho',i{+}1,?))\ \frac{\tau_1\ \ldots\ \tau_i\ c' \Rightarrow ?}{c \Rightarrow ?} \quad \begin{array}{l}\rho' \sim_i \rho\\ R(\rho', i) = r\\ C(\rho', i+1) = c'\end{array}$$

$$\text{(TR-5)}\quad \text{(pev}_?(\rho,i,?))\ \frac{\tau_1\ \ldots\ \tau_{i-1}\ \tau_i}{c \Rightarrow ?} \longrightarrow_{\mathcal{R}} \text{(pev}_?(\rho,i,r_?))\ \frac{\tau_1\ \ldots\ \tau_{i-1}\ \tau_i'}{c \Rightarrow ?} \quad \tau_i \longrightarrow_{\mathcal{R}} \tau_i'$$

Fig. 3. Transition relation between partial evaluation trees.

$$\frac{}{(\lambda x.x)\ n \Rightarrow ?} \longrightarrow_{\mathcal{R}} \frac{\lambda x.x \Rightarrow ?}{(\lambda x.x)\ n \Rightarrow ?} \longrightarrow_{\mathcal{R}} \frac{\lambda x.x \Rightarrow \lambda x.x}{(\lambda x.x)\ n \Rightarrow ?}$$

$$\longrightarrow_{\mathcal{R}} \frac{\lambda x.x \Rightarrow \lambda x.x \quad n \Rightarrow ?}{(\lambda x.x)\ n \Rightarrow ?} \longrightarrow_{\mathcal{R}} \frac{\lambda x.x \Rightarrow \lambda x.x \quad n \Rightarrow n}{(\lambda x.x)\ n \Rightarrow ?}$$

$$\longrightarrow_{\mathcal{R}} \frac{\lambda x.x \Rightarrow \lambda x.x \quad n \Rightarrow n \quad n \Rightarrow ?}{(\lambda x.x)\ n \Rightarrow ?} \longrightarrow_{\mathcal{R}} \frac{\lambda x.x \Rightarrow \lambda x.x \quad n \Rightarrow n \quad n \Rightarrow n}{(\lambda x.x)\ n \Rightarrow ?}$$

$$\longrightarrow_{\mathcal{R}} \frac{\lambda x.x \Rightarrow \lambda x.x \quad n \Rightarrow n \quad n \Rightarrow n}{(\lambda x.x)\ n \Rightarrow n}$$

Fig. 4. The evaluation of $(\lambda x.x)\ n$ using $\longrightarrow_{\mathcal{R}}$ for rules in Fig. 1.

- If the last applied rule is a propagation rule $\text{pev}_?(\rho, i, ?)$, then we simply propagate the step made by τ_i (the last subtree), which is necessarily incomplete. After the step, τ_i' may be complete, hence the last applied rule is $\text{pev}_?(\rho, i, r_?)$.

In Fig. 4 we report an example of evaluation of a term according to rules in Fig. 1, using partial evaluation trees and $\longrightarrow_{\mathcal{R}}$.

As mentioned above, the definition of $\longrightarrow_{\mathcal{R}}$ given in Fig. 3 nicely models as a transition system an interpreter driven by the big-step rules. In other words, the one-step transition relation between finite partial evaluation trees specifies an algorithm of incremental evaluation.[5] On the other hand, also the partial order relation $\sqsubseteq$ (cf. Definition 4.6) models a refinement relation between finite partial evauation trees, even if in a more abstract way. The next proposition formally proves that these two descriptions agree, namely, $\sqsubseteq$ is indeed an abstraction of $\longrightarrow_{\mathcal{R}}$.

Proposition 4.13. *Let τ and τ' be finite partial evaluation trees, then the following hold:*

(1) if $\tau \longrightarrow_{\mathcal{R}} \tau'$ then $\tau \sqsubset \tau'$, and
(2) if $\tau \sqsubseteq \tau'$ then $\tau \longrightarrow_{\mathcal{R}}^{\star} \tau'$.

Proof. Point 1 can be easily proved by induction on the definition of $\longrightarrow_{\mathcal{R}}$. The proof of point 2 is by induction on τ', denote by *IH* the induction hypothesis. This is possible as τ' is finite by hypothesis. We can assume $R_?(\mathrm{r}(\tau)) = ?$,

[5]Non-determinism can only be caused by intrinsic non-determinism of the big-step semantics, if any.

since in the other case, by Proposition 4.7 (1), we have $\tau = \tau'$, hence the thesis is trivial. We can further assume $R_?(\mathsf{r}(\tau')) = ?$, since, if $\tau' = \dfrac{\tau'_1 \quad \cdots \quad \tau'_n}{c \Rightarrow r}$, then we always have $\tau'' = \dfrac{\tau'_1 \quad \cdots \quad \tau'_n}{c \Rightarrow ?} \longrightarrow_{\mathcal{R}} \tau'$ and $\tau \sqsubseteq \tau''$, because $\tau \sqsubseteq \tau'$ and we have $\mathsf{dom}(\tau') = \mathsf{dom}(\tau'')$, $\tau'(\alpha) = \tau''(\alpha)$ for all $\alpha \neq \varepsilon$, $C(\mathsf{r}(\tau')) = C(\mathsf{r}(\tau''))$ and $R_?(\mathsf{r}(\tau)) = ?$. Now, if $\tau' = \dfrac{}{c \Rightarrow ?}$ (base case), then, since $\mathsf{dom}(\tau) \subseteq \mathsf{dom}(\tau')$ and $C(\mathsf{r}(\tau)) = C(\mathsf{r}(\tau'))$ by definition of $\sqsubseteq$, we have $\tau = \tau'$, hence the thesis is trivial.

Let us assume $\tau = \dfrac{\tau_1 \quad \ldots \tau_k}{c \Rightarrow ?}$ and $\tau' = \dfrac{\tau'_1 \quad \cdots \quad \tau'_i}{c' \Rightarrow ?}$, with, necessarily, $k \leq i$ and $c = c'$ by definition of $\sqsubseteq$. We have $\tau_h \sqsubseteq \tau'_h$, for all $h \leq k$, and by Proposition 4.4 (2), at most τ_k is incomplete, that is, for all $h < k$, τ_h is complete, namely, $R_?(\mathsf{r}(\tau_h)) \in R$, thus, by definition of $\sqsubseteq$, we have $\tau_h = \tau'_h$. Furthermore, since $\tau_k \sqsubseteq \tau'_k$, by IH, we get $\tau_k \longrightarrow^{\star}_{\mathcal{R}} \tau'_k$, hence $\tau \longrightarrow^{\star}_{\mathcal{R}} \tau'' = \dfrac{\tau'_1 \quad \cdots \quad \tau'_k}{c \Rightarrow ?} \sqsubseteq \tau'$. We now show, concluding the proof, by arithmetic induction on $i - k$, that $\tau'' \longrightarrow^{\star}_{\mathcal{R}} \tau'$. If $i - k = 0$, hence $i = k$, we have $\tau'' = \tau'$, hence the thesis is immediate. If $i - k > 0$, hence $i > k$, setting $c'' = C(\mathsf{r}(\tau'_{k+1}))$, by IH, we get $\dfrac{}{c'' \Rightarrow ?} \longrightarrow^{\star}_{\mathcal{R}} \tau'_{k+1}$; moreover, again by Proposition 4.4 (2), we have $R_?(\mathsf{r}(\tau'_k)) \in R$, hence we get

$$\tau'' \longrightarrow_{\mathcal{R}} \frac{\tau'_1 \quad \ldots \quad \tau'_k \quad c'' \Rightarrow ?}{c \Rightarrow ?} \longrightarrow^{\star}_{\mathcal{R}} \frac{\tau'_1 \quad \ldots \quad \tau'_k \quad \tau'_{k+1}}{c \Rightarrow ?} = \hat{\tau}$$

Finally, by arithmetic induction hypothesis, we get $\hat{\tau} \longrightarrow^{\star}_{\mathcal{R}} \tau'$, as needed. □

We conclude this section by showing that the transition relation $\longrightarrow_{\mathcal{R}}$ agrees with the semantic relation (inductively) defined by $\mathcal{R}$, namely, the semantic relation captures exactly successful terminating computations in $\longrightarrow_{\mathcal{R}}$.

Theorem 4.14. *$\mathcal{R} \vdash_{\mu} c \Rightarrow r$ iff $\dfrac{}{c \Rightarrow ?} \longrightarrow^{\star}_{\mathcal{R}} \tau$, where $\mathsf{r}(\tau) = c \Rightarrow r$.*

Proof. $\mathcal{R} \vdash_{\mu} c \Rightarrow r$ implies $\dfrac{}{c \Rightarrow ?} \longrightarrow^{\star}_{\mathcal{R}} \tau$ where $\mathsf{r}(\tau) = c \Rightarrow r$. By definition, if $\mathcal{R} \vdash_{\mu} c \Rightarrow r$ holds, then there is a finite evaluation tree τ in $\mathcal{R}$ such that $\mathsf{r}(\tau) = c \Rightarrow r$. Since $\mathcal{R} \subseteq \mathcal{R}_?$ by Definition 4.1, τ is a (complete) partial evaluation tree as well; furthermore, $\dfrac{}{c \Rightarrow ?} \sqsubseteq \tau$, hence, by Proposition 4.13 (2), we get the thesis.

$\dfrac{}{c \Rightarrow ?} \longrightarrow^{\star}_{\mathcal{R}} \tau$ where $\mathsf{r}(\tau) = c \Rightarrow r$ implies $\mathcal{R} \vdash_{\mu} c \Rightarrow r$. Since $\mathsf{r}(\tau) = c \Rightarrow r$, by Corollary 4.5, τ is complete, hence, it is an evaluation tree in $\mathcal{R}$, thus $\mathcal{R} \vdash_{\mu} c \Rightarrow r$ holds. □

5 EXTENDED BIG-STEP SEMANTICS: TWO CONSTRUCTIONS

In Section 4, we have just shown that, given a big-step semantics as in Definition 3.1, it is possible to define computations in such semantics, by deriving a transition relation which formally models the evaluation algorithm guided by the rules. In this way, we are able to distinguish stuck and non-terminating computations as in standard small-step semantics. This, in a sense, shows that such a distinction is *implicit* in a big-step semantics.

In this section, we aim at showing that we can make such distinction explicit directly by a big-step semantics, without introducing any transition relation modelling single computation steps. To this end, we describe two constructions that, starting from a big-step semantics, yield extended ones where non-terminating and stuck computations are explicitly distinguished. These two constructions are in some sense dual to each other, because one explicitly models non-termination, while the other one explicitly models stuckness, and they are based on well-know ideas: divergence is modelled by *traces*, as suggested by Leroy and Grall [36], while stuckness by an additional special result, as described, for instance, by Pierce [44]. The novel contribution is that, thanks to the general definition of big-step semantics in Section 3 (cf. Definition 3.1), we can provide *general* constructions working on an arbitrary big-step semantics, rather than discussing specific examples, as it is customary in the literature.

In the following, we assume a big-step semantics $\langle C, R, \mathcal{R}\rangle$.

5.1 Adding traces

The set of *traces* in the big-step semantics is the set C^∞ of finite and infinite sequences of configurations. Finite traces are ranged over by t, while infinite traces by σ.

The judgement of trace semantics has shape $c \Rightarrow_{\mathsf{tr}} r_{\mathsf{tr}}$, where $r_{\mathsf{tr}} \in Tr_R^C = (C^\star \times R) + C^\omega$, that is, r_{tr} is either a pair $\langle t, r\rangle$ of a finite trace and a result, modelling a converging computation, or an infinite trace σ, modelling divergence. Intuitively, traces t keep track of all the configurations visited during the evaluation, starting from c itself. To define the trace semantics, we construct, starting from $\mathcal{R}$, a new set of rules $\mathcal{R}_{\mathsf{tr}}$ as follows:

Definition 5.1 (Rules for traces). The set of rules $\mathcal{R}_{\mathsf{tr}}$ consists of the following rules:

finite trace rules For each $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ in $\mathcal{R}$ and finite traces $t_1, \dots, t_n \in C^\star$, define rule $\mathsf{trace}(\rho,\ t_1, \dots, t_n)$ as

$$\frac{C(j_1) \Rightarrow_{\mathsf{tr}} \langle t_1, R(j_1)\rangle \quad \dots \quad C(j_n) \Rightarrow_{\mathsf{tr}} \langle t_n, R(j_n)\rangle}{c \Rightarrow_{\mathsf{tr}} \langle c t_1 \cdots t_n, r\rangle}$$

infinite trace rules For each $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ in $\mathcal{R}$, index $i \in 1..n$, finite traces $t_1, \dots, t_{i-1} \in C^\star$, and infinite trace $\sigma \in C^\omega$, define rule $\mathsf{trace}_\infty(\rho,\ i,\ t_1, \dots, t_{i-1},\ \sigma)$ as follows:

$$\frac{C(j_1) \Rightarrow_{\mathsf{tr}} \langle t_1, R(j_1)\rangle \quad \dots \quad C(j_{i-1}) \Rightarrow_{\mathsf{tr}} \langle t_{i-1}, R(j_{i-1})\rangle \quad C(j_i) \Rightarrow_{\mathsf{tr}} \sigma}{c \Rightarrow_{\mathsf{tr}} c t_1 \cdots t_{i-1} \sigma}$$

Finite trace rules enrich big-step rules in $\mathcal{R}$ by finite traces, thus modelling computations converging to a final result. On the other hand, infinite trace rules handle non-termination, modelled by infinite traces: they propagate divergence, that is, if a configuration in the premises of a rule in $\mathcal{R}$ diverges, namely, it evaluates to an infinite trace, then the subsequent premises are ignored and the configuration in the conclusion diverges as well. Note that all these rules have a non-empty trace in the conclusion, hence only non-empty traces are derivable by such rules. Finally, observe that the triple $\langle C, Tr_R^C, \mathcal{R}_{\mathsf{tr}}\rangle$ is a big-step semantics according to Definition 3.1.

The standard inductive interpretation of big-step rules is not enough in this setting: it can only derive judgements of shape $c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$, because there is no axiom introducing infinite traces, hence they cannot be derived by finite derivations. In other words, the inductive interpretation of $\mathcal{R}_{\mathsf{tr}}$ can only capture converging computations. To properly handle divergence, we have to interpret rules *coinductively*, namely, allowing both finite and infinite derivations. Then, we will write $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} r_{\mathsf{tr}}$ to say that $c \Rightarrow_{\mathsf{tr}} r_{\mathsf{tr}}$ is coinductively derivable by rules in $\mathcal{R}_{\mathsf{tr}}$. It is important to note the following proposition, stating that enabling infinite derivations does not affect the semantics of converging computations.

Lemma 5.2. $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$ *iff* $\mathcal{R}_{\mathsf{tr}} \vdash_\mu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$.

Proof. The right-to-left implication is trivial, because the inductive interpretation is always included in the coinductive one. The proof of the other direction is by induction on the length of t, which is a finite trace. By hypothesis, we know that $c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$ is derivable by a (possibly infinite) derivation and, by Definition 5.1, we know that the last applied rule ρ^{tr} has shape $\mathsf{trace}(\rho,\ t_1, \dots, t_n)$, hence $t = c t_1 \cdots t_n$. If $|t| = 1$, then $t = c$, and so $n = 0$, that is, $\rho = \mathsf{rule}(\varepsilon,\ c,\ r)$, because only non-empty traces are derivable, hence $\mathcal{R}_{\mathsf{tr}} \vdash_\mu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$ holds by ρ^{tr}. If $|t| > 0$, then, for all $i \in 1..n$, $|t_i| < |t|$, hence, by induction hypothesis, we get $\mathcal{R}_{\mathsf{tr}} \vdash_\mu C(\rho, i) \Rightarrow_{\mathsf{tr}} \langle t_i, R(\rho, i)\rangle$, and so $\mathcal{R}_{\mathsf{tr}} \vdash_\mu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$ holds by ρ^{tr}. □

$$\text{(APP-TR)}\ \frac{e_1 \Rightarrow_{\mathsf{tr}} \langle t_1, \lambda x.e\rangle \quad e_2 \Rightarrow_{\mathsf{tr}} \langle t_2, v_2\rangle \quad e[v_2/x] \Rightarrow_{\mathsf{tr}} \langle t, v\rangle}{e_1\ e_2 \Rightarrow_{\mathsf{tr}} \langle (e_1\ e_2) t_1 t_2 t, v\rangle}$$

$$\text{(DIV-APP-1)}\ \frac{e_1 \Rightarrow_{\mathsf{tr}} \sigma}{e_1\ e_2 \Rightarrow_{\mathsf{tr}} (e_1\ e_2)\sigma} \qquad \text{(DIV-APP-2)}\ \frac{e_1 \Rightarrow_{\mathsf{tr}} \langle t_1, \lambda x.e\rangle \quad e_2 \Rightarrow_{\mathsf{tr}} \sigma}{e_1\ e_2 \Rightarrow_{\mathsf{tr}} (e_1\ e_2) t_1 \sigma}$$

$$\text{(DIV-APP-3)}\ \frac{e_1 \Rightarrow_{\mathsf{tr}} \langle t_1, \lambda x.e\rangle \quad e_2 \Rightarrow_{\mathsf{tr}} \langle t_2, v_2\rangle \quad e[v_2/x] \Rightarrow_{\mathsf{tr}} \sigma}{e_1\ e_2 \Rightarrow_{\mathsf{tr}} (e_1\ e_2) t_1 t_2 \sigma}$$

Fig. 5. Trace semantics for application

Note that, following the same inductive strategy as the above proof, we can prove that actually a derivation for a judgement of shape $c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$ is necessarily finite. This is essentially due to the fact that rules are *productive*, meaning that the trace in the conclusion is always strictly larger than those in the premises.

We show in Fig. 5 the rules obtained by applying Definition 5.1, starting from meta-rule (APP) of the example in Fig. 1 (for the other meta-rules the outcome is analogous).

For instance, set $\omega = \lambda x.x\,x$, hence $\Omega = \omega\,\omega$ (cf. page 11), and σ_Ω the infinite trace $\Omega\omega\omega\Omega\omega\omega\ldots$, it is easy to see that the judgment $\Omega \Rightarrow_{\mathsf{tr}} t_\Omega$ can be derived by the following infinite derivation:[6]

$$\frac{\dfrac{}{\omega \Rightarrow_{\mathsf{tr}} \langle \omega, \omega\rangle} \quad \dfrac{}{\omega \Rightarrow_{\mathsf{tr}} \langle \omega, \omega\rangle} \quad \dfrac{\vdots}{\Omega = (x\,x)[\omega/x] \Rightarrow_{\mathsf{tr}} \sigma_\Omega}}{\Omega \Rightarrow \Omega\omega\omega\sigma_\Omega = \sigma_\Omega}$$

Note that *only* the judgment $\Omega \Rightarrow_{\mathsf{tr}} \sigma_\Omega$ can be derived, that is, the trace semantics of Ω is uniquely determined to be σ_Ω, since the infinite derivation forces the equation $\sigma_\Omega = \Omega\omega\omega\sigma_\Omega$.

To check that the construction in Definition 5.1 is a correct extension of the given big-step semantics, we have to show it is *conservative*, in the sense that it does not affect the semantics of converging computations, as formally stated below.

THEOREM 5.3. *$\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$ for some $t \in C^\star$ iff $\mathcal{R} \vdash_\mu c \Rightarrow r$.*

PROOF. By Lemma 5.2, we know that $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$ iff $\mathcal{R}_{\mathsf{tr}} \vdash_\mu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$. Then, the thesis follows by proving $\mathcal{R}_{\mathsf{tr}} \vdash_\mu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$, for some $t \in C^\star$, iff $\mathcal{R} \vdash_\mu c \Rightarrow r$, by a straightforward induction on rules. □

We conclude this subsection by showing a coinductive proof principle associated with trace semantics, which allows us to prove that a predicate on configurations ensures the existence of a non-terminating computation.

LEMMA 5.4. *Let $\mathcal{S} \subseteq C$ be a set. If, for all $c \in \mathcal{S}$, there are $\rho = \mathsf{rule}(j_1 \ldots j_n,\ c,\ r) \in \mathcal{R}$ and $i \in 1..n$ such that*

(1) for all $k < i$, $\mathcal{R} \vdash_\mu j_k$, and
(2) $C(j_i) \in \mathcal{S}$

then, for all $c \in \mathcal{S}$, there exists $\sigma \in C^\omega$ such that $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \sigma$.

PROOF. First of all, for each $c \in \mathcal{S}$, we construct a trace $\sigma_c \in C^\omega$, which will be the candidate trace to prove the thesis. By hypothesis (Item 1), there is a rule $\rho_c = \mathsf{rule}(j_1^c \ldots j_{n_c}^c,\ c,\ r_c)$ and an index $i_c \in 1..n_c$ such that, for all $k < i_c$, we have $\mathcal{R} \vdash_\mu j_k^c$. Therefore, by Theorem 5.3, there are finite traces $t_1^c, \ldots, t_{i_c-1}^c \in C^\star$ such that for all $k < i_c$ we have

[6] To help the reader, we add equivalent expressions with a grey background.

$\mathcal{R}_{\mathsf{tr}} \vdash_\nu C(j^c_k) \Rightarrow_{\mathsf{tr}} \langle t^c_k, R(j^c_k)\rangle$, and, in addition (Item 2), we know that $C(j^c_{i_c}) \in \mathcal{S}$. Then, for each $c \in \mathcal{S}$, we can introduce a variable X_c and define an equation $X_c = c \cdot t^c_1 \cdot \dots \cdot t^c_{i_c-1} \cdot X_{C(j^c_{i_c})}$. The set of all such equations is a guarded system of equations, which thus has a unique solution, namely, a function $s : \mathcal{S} \to C^\omega$ such that, for each $c \in \mathcal{S}$ we have $s(c) = c \cdot t^c_1 \cdot \dots \cdot t^c_{i_c-1} \cdot s(C(j^c_{i_c}))$.[7]

We now have to prove that, for all $c \in \mathcal{S}$, we have $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} s(c)$. To this end, consider the set $\mathcal{S}' = \{\langle c, s(c)\rangle \mid c \in \mathcal{S}\} \cup \{\langle c, \langle t, r\rangle\rangle \mid \mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle\}$, then the proof is by coinduction. Let $\langle c, r_{\mathsf{tr}}\rangle \in \mathcal{S}'$, then we have to find a rule $\langle j_1 \dots j_n, c \Rightarrow_{\mathsf{tr}} r_{\mathsf{tr}}\rangle \in \mathcal{R}_{\mathsf{tr}}$ such that, for all $k \in 1..n$, $\langle C(j_k), Tr^C_R(j_k)\rangle \in \mathcal{S}'$. We have two cases:

- if $r_{\mathsf{tr}} = s(c)$, then the needed rule is $\mathsf{trace}_\infty(\rho_c,\ i_c,\ t^c_1, \dots, t^c_{i_c-1},\ s(C(j^c_{i_c})))$, and
- if $r_{\mathsf{tr}} = \langle t, r\rangle$, then $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$, by construction of $\mathcal{S}'$, hence $c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$ is the conclusion of a finite trace rule, where all premises are still derivable, thus in $\mathcal{S}'$ by construction.

□

5.2 Adding wrong

A well-known technique [1, 44] to distinguish between stuck and diverging computations, in a sense "dual" to the previous one, is to add a special result wrong, so that $c \Rightarrow \mathsf{wrong}$ means that the evaluation of c goes stuck.

In this case, defining a general and "automatic" version of the construction, starting from an arbitrary big-step semantics $\langle C, R, \mathcal{R}\rangle$, is a non-trivial problem. Our solution is based on the equivalence on rules defined in Definition 4.12 (equality up to an index), which allows us to define when wrong can be introduced.

The extended judgement has shape $c \Rightarrow r_{\mathsf{wr}}$ where $r_{\mathsf{wr}} \in R_{\mathsf{wr}} = R + \{\mathsf{wrong}\}$, that is, it is either a result or an error. To define the extended semantics, we construct, starting from $\mathcal{R}$, an extended set of rules $\mathcal{R}_{\mathsf{wr}}$ as follows:

Definition 5.5 (Rules for wrong). The set of rules $\mathcal{R}_{\mathsf{wr}}$ is obtained by adding to $\mathcal{R}$ the following rules:

wrong configuration rules For each configuration $c \in C$ such that there is no rule ρ in $\mathcal{R}$ with $C(\rho) = c$, define rule $\mathsf{wrong}(c)$ as $\dfrac{}{c \Rightarrow \mathsf{wrong}}$.

wrong result rules For each rule $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ in $\mathcal{R}$, index $i \in 1..n$, and result $r' \in R$, if, for all rules ρ' such that $\rho \sim_i \rho'$, $R(\rho', i) \neq r'$, then define rule $\mathsf{wrong}(\rho,\ i,\ r')$ as

$$\frac{j_1 \quad \dots \quad j_{i-1} \quad C(j_i) \Rightarrow r'}{c \Rightarrow \mathsf{wrong}}$$

wrong propagation rules These rules propagate wrong analogously to those for divergence propagation: For each rule $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ in $\mathcal{R}$ and index $i \in 1..n$, define rule $\mathsf{prop}(\rho, i, \mathsf{wrong})$ as

$$\frac{j_1 \quad \dots \quad j_{i-1} \quad C(j_i) \Rightarrow \mathsf{wrong}}{c \Rightarrow \mathsf{wrong}}$$

Wrong configurations rules simply say that, if there is no rule for a given configuration, then we can derive wrong. Wrong result rules, instead, derive wrong whenever the configuration in a premise of a rule reduces to a result which is not admitted in such (and any equivalent) rule. We also call these two kinds of rules wrong introduction rules, as they introduce wrong in the conclusion without having it in the premises. Finally, wrong propagation rules say that, if a configuration in a premise of some rule in $\mathcal{R}$ goes wrong, then the subsequent premises are ignored and the configuration in the conclusion goes wrong as well. Note that $\langle C, R_{\mathsf{wr}}, \mathcal{R}_{\mathsf{wr}}\rangle$ is a big-step semantics according to Definition 3.1.

[7] This argument can be made more precise using coalgebras [50], in particular the fact that $\mathcal{S}$ and C^ω carry, respectively, a coalgebra and a corecursive algebra [16] structure for the functor $X \mapsto C^\star \times X$.

$$\text{(WRONG-APP)}\ \frac{e_1 \Rightarrow n}{e_1\ e_2 \Rightarrow \mathsf{wrong}} \qquad \text{(WRONG-SUCC)}\ \frac{e \Rightarrow \lambda x.e}{\mathsf{succ}\ e \Rightarrow \mathsf{wrong}}$$

$$\text{(PROP-APP-1)}\ \frac{e_1 \Rightarrow \mathsf{wrong}}{e_1\ e_2 \Rightarrow \mathsf{wrong}} \qquad \text{(PROP-APP-2)}\ \frac{e_1 \Rightarrow \lambda x.e \quad e_2 \Rightarrow \mathsf{wrong}}{e_1\ e_2 \Rightarrow \mathsf{wrong}}$$

$$\text{(PROP-APP-3)}\ \frac{e_1 \Rightarrow \lambda x.e \quad e_2 \Rightarrow v_2 \quad e[v_2/x] \Rightarrow \mathsf{wrong}}{e_1\ e_2 \Rightarrow \mathsf{wrong}} \qquad \text{(PROP-SUCC)}\ \frac{e \Rightarrow \mathsf{wrong}}{\mathsf{succ}\ e \Rightarrow \mathsf{wrong}}$$

Fig. 6. Semantics with wrong for application and successor

In this case, the standard inductive interpretation is enough to get the correct semantics, because, intuitively, an error, if any, occurs after a finite number of steps. Then, we write $\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow r_{\mathsf{wr}}$ when the judgment $c \Rightarrow r_{\mathsf{wr}}$ is inductively derivable by rules in $\mathcal{R}_{\mathsf{wr}}$.

We show in Fig. 6 the meta-rules for wrong introduction and propagation constructed starting from those for application and successor in Fig. 1.

For instance, rule (WRONG-APP) is introduced since in the original semantics there is rule (APP) with $e_1\ e_2$ in the conclusion and e_1 in the first premise, but there is no equivalent rule (that is, with $e_1\ e_2$ in the conclusion and e_1 in the first premise) such that the result in the first premise is n. Intuitively, this means that n is a wrong result for the evaluation of the first argument of an application.

Like the previous construction, the wrong construction is a correct extension of $\mathcal{R}$, namely, it is conservative.

Theorem 5.6. *$\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow r$ iff $\mathcal{R} \vdash_\mu c \Rightarrow r$.*

Proof. The right-to-left implication is trivial, as $\mathcal{R} \subseteq \mathcal{R}_{\mathsf{wr}}$ by Definition 5.5. The proof of the other direction is by induction on rules in $\mathcal{R}_{\mathsf{wr}}$. The only relevant cases are rules in $\mathcal{R}$, because rules in $\mathcal{R}_{\mathsf{wr}} \setminus \mathcal{R}$ allow only the derivation of judgements of shape $c \Rightarrow \mathsf{wrong}$. Hence, the thesis is immediate. □

5.3 Correctness of constructions

We now prove correctness of the trace and wrong constructions, by showing they capture diverging and stuck computations, respectively, as defined by the transition relation $\longrightarrow_{\mathcal{R}}$ introduced in Section 4.2. This provides us a coherence result for our approach.

First of all, note that both constructions correctly capture converging computations, because, if restricted to such computations, by Theorems 5.3 and 5.6, the constructions are both equivalent to the original big-step semantics. Hence, in the following, we focus only on diverging and stuck computations, respectively.

Correctness of $\mathcal{R}_{\mathsf{tr}}$. Given a partial evaluation tree τ, we write $\tau \longrightarrow^{\omega}_{\mathcal{R}}$ meaning that there is an infinite sequence of $\longrightarrow_{\mathcal{R}}$-steps starting from τ. Then, the theorem we want to prove is the following:

Theorem 5.7. *$\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \sigma$, for some $\sigma \in C^\omega$, iff* $\dfrac{}{c \Rightarrow ?} \longrightarrow^{\omega}_{\mathcal{R}}$.

To prove this result, we need to relate evaluation trees (a.k.a. derivations) in $\mathcal{R}_{\mathsf{tr}}$ (cf. Definition 5.1) to partial evaluation trees in $\mathcal{R}$ (cf. Definition 4.3). To this end, we define a function $u_? : Tr^C_R \to R_?$, which essentially forgets traces, as follows: $u_?(\langle t, r\rangle) = r$ and $u_?(\sigma) = ?$. We can extend this function to judgements, mapping $c \Rightarrow_{\mathsf{tr}} r_{\mathsf{tr}}$ to $c \Rightarrow u_?(r_{\mathsf{tr}})$, and to rules, mapping $\mathsf{trace}(\rho,\ t_1, \ldots, t_n)$ to ρ and $\mathsf{trace}_\infty(\rho,\ i,\ t_1, \ldots, t_{i-1},\ \sigma)$ to $\mathsf{pev}_?(\rho, i, ?)$. Finally, we get a function erase that transforms an evaluation tree τ^{tr} in $\mathcal{R}_{\mathsf{tr}}$ into a partial evaluation tree, defined by $\mathsf{erase}(\tau^{\mathsf{tr}}) = u_? \circ \tau^{\mathsf{tr}}$, that is, relying on the fact that a tree is a (partial) function, we postcompose τ^{tr} with $u_?$; in other words, this means that we

apply $u_?$ to all judgements labeling a node in τ^{tr}, thus erasing traces. Since $u_?$ transforms rules in $\mathcal{R}_{\mathsf{tr}}$ into rules in $\mathcal{R}_?$, $\mathsf{erase}(\tau^{\mathsf{tr}})$ is indeed a partial evaluation tree and the following equalities between trees hold:

$$\mathsf{erase}\left(\begin{array}{c}(\mathsf{trace}(\rho, t_1, \ldots, t_n))\\ \hline c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle\end{array}\begin{array}{c}\tau_1^{\mathsf{tr}} \quad \ldots \quad \tau_n^{\mathsf{tr}}\end{array}\right) = \begin{array}{c}(\rho)\\ \hline c \Rightarrow r\end{array}\begin{array}{c}\mathsf{erase}\left(\tau_1^{\mathsf{tr}}\right) \quad \ldots \quad \mathsf{erase}\left(\tau_n^{\mathsf{tr}}\right)\end{array}$$

$$\mathsf{erase}\left(\begin{array}{c}(\mathsf{trace}_\infty(\rho, i, t_1, \ldots, t_{i-1}, \sigma))\\ \hline c \Rightarrow_{\mathsf{tr}} \sigma'\end{array}\begin{array}{c}\tau_1^{\mathsf{tr}} \quad \ldots \quad \tau_i^{\mathsf{tr}}\end{array}\right) = \begin{array}{c}(\mathsf{pev}_?(\rho, i, ?))\\ \hline c \Rightarrow ?\end{array}\begin{array}{c}\mathsf{erase}\left(\tau_1^{\mathsf{tr}}\right) \quad \ldots \quad \mathsf{erase}\left(\tau_i^{\mathsf{tr}}\right)\end{array}$$

Note that, by construction, $\mathsf{dom}(\tau^{\mathsf{tr}}) = \mathsf{dom}(\mathsf{erase}\left(\tau^{\mathsf{tr}}\right))$, hence, τ^{tr} is finite iff $\mathsf{erase}\left(\tau^{\mathsf{tr}}\right)$ is finite and τ^{tr} is infinite iff $\mathsf{erase}\left(\tau^{\mathsf{tr}}\right)$ is infinite. Furthermore, since, as we have already observed, τ^{tr} is finite iff $\mathsf{r}(\tau^{\mathsf{tr}}) = c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$, we have that $\mathsf{erase}\left(\tau^{\mathsf{tr}}\right)$ is complete iff τ^{tr} is finite and $\mathsf{erase}\left(\tau^{\mathsf{tr}}\right)$ is well-formed iff τ^{tr} is infinite (cf. Definition 4.9).

Lemma 5.8. *If $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} r_{\mathsf{tr}}$ holds by an infinite evaluation tree τ^{tr}, then there is a sequence $(\tau_n)_{n\in\mathbb{N}}$ such that $\tau_n \longrightarrow_{\mathcal{R}} \tau_{n+1}$ for all $n \in \mathbb{N}$, $\tau_0 = \dfrac{}{c \Rightarrow ?}$, and $\bigsqcup \tau_n = \mathsf{erase}\left(\tau^{\mathsf{tr}}\right)$.*

Proof. Since τ^{tr} is infinite, $\mathsf{erase}\left(\tau^{\mathsf{tr}}\right) = \tau$ is a well-formed infinite partial evaluation treee and, by Proposition 4.11 (2), there is a strictly increasing sequence $(\tau'_n)_{n\in\mathbb{N}}$ of finite partial evaluation trees such that $\bigsqcup \tau'_n = \tau$ and $\tau'_0 = \dfrac{}{c \Rightarrow ?}$. By Proposition 4.13 (2), since for all $n \in \mathbb{N}$ we have $\tau'_n \sqsubset \tau'_{n+1}$, we get $\tau'_n \longrightarrow^\star_{\mathcal{R}} \tau'_{n+1}$, and, since $\tau'_n \neq \tau'_{n+1}$, this sequence of steps is not empty. Hence, we can construct a sequence $(\tau_n)_{n\in\mathbb{N}}$ such that $\tau_0 = \tau'_0 = \dfrac{}{c \Rightarrow ?}$, $\tau_n \longrightarrow_{\mathcal{R}} \tau_{n+1}$ and $\bigsqcup \tau_n = \tau$, as needed. □

Lemma 5.9. *Let τ be a well-formed infinite partial evaluation tree with $\mathsf{r}(\tau) = c \Rightarrow ?$. Then, $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \sigma$ holds for some $\sigma \in C^\omega$.*

Proof. The thesis follows from Lemma 5.4, applied to the set $\mathcal{S} \subseteq C$ defined as follows: $c \in \mathcal{S}$ iff $C(\mathsf{r}(\tau)) = c$, for some infinite well-formed partial evaluation tree τ. Let $c \in \mathcal{S}$, then $c = C(\mathsf{r}(\tau))$ and the last applied rule in τ is $\mathsf{pev}_?(\rho, i, ?)$, for some $\rho = \mathsf{rule}(j_1 \ldots j_n,\ c,\ r)$ in $\mathcal{R}$. Then, we have $\mathcal{R} \vdash_\mu j_k$, for all $k < i$ and $C(j_i) = C(\mathsf{r}(\tau_{|_i}))$ and $\tau_{|_i}$ is an infinite well-formed partial evaluation tree. Therefore, $C(j_i) \in \mathcal{S}$, and so the hypotheses of Lemma 5.4 are satisfied. □

Proof of Theorem 5.7. $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \sigma$ for some $\sigma \in C^\omega$ implies $\dfrac{}{c \Rightarrow ?} \longrightarrow^\omega_{\mathcal{R}}$. Since $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \sigma$ holds and σ is infinite, by (a consequence of) Lemma 5.2, there is an infinite evaluation tree τ^{tr} in $\mathcal{R}_{\mathsf{tr}}$ such that $\mathsf{r}(\tau^{\mathsf{tr}}) = c \Rightarrow_{\mathsf{tr}} \sigma$. Then, by Lemma 5.8 we get the thesis.

$\dfrac{}{c \Rightarrow ?} \longrightarrow^\omega_{\mathcal{R}}$ implies $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \sigma$ for some $\sigma \in C^\omega$. By definition of $\longrightarrow^\omega_{\mathcal{R}}$, there is an infinite sequence $(\tau_n)_{n\in\mathbb{N}}$ such that $\tau_0 = \dfrac{}{c \Rightarrow ?}$ and, for all $n \in \mathbb{N}$, $\tau_n \longrightarrow_{\mathcal{R}} \tau_{n+1}$, hence, by Proposition 4.13 (1), we get $\tau_n \sqsubset \tau_{n+1}$. By Proposition 4.11 (1), we have that $\tau = \bigsqcup \tau_n$ is a well-formed infinite partial evaluation tree, hence we get the thesis by Lemma 5.9. □

Correctness of $\mathcal{R}_{\mathsf{wr}}$. We now show that the construction in Section 5.2 correctly models stuck computation in $\longrightarrow_{\mathcal{R}}$.

The proof relies on the following lemma. We say that a (finite) partial evaluation tree τ is *irreducible* if there is no τ' such that $\tau \longrightarrow_{\mathcal{R}} \tau'$, and it is *stuck* if it is irreducible and $R_?(\mathsf{r}(\tau)) = ?$. Note that, by Proposition 4.7 (1) and Proposition 4.13 (1), a complete partial evaluation tree τ is irreducible.

Lemma 5.10. *If τ is a stuck partial evaluation tree with $\mathsf{r}(\tau) = c \Rightarrow ?$, then $\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow$ wrong holds.*

Proof. The proof is by induction on τ, splitting cases on the last applied rule.

Case: $\mathsf{ax}_?(c)$ Since τ is stuck, by definition of $\longrightarrow_{\mathcal{R}}$ (cf. Fig. 3 first and second clauses), there is no rule $\rho \in \mathcal{R}$ such that $C(\rho) = c$, hence $\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow \mathsf{wrong}$ holds, by applying $\mathsf{wrong}(c)$.

Case: $\mathsf{pev}_?(\rho, i, r)$ Suppose $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r')$ and $i \in 1..n$, by hypothesis, for all $k < i$, $\tau_{|_k}$ is a complete partial evaluation tree of j_k, hence we know that $\mathcal{R} \vdash_\mu j_k$ holds. Since τ is stuck, by definition of $\longrightarrow_{\mathcal{R}}$ (cf. Fig. 3 third and fourth clauses), there is no rule $\rho' \sim_i \rho$ with $R(\rho', i) = r$, hence $\mathsf{wrong}(\rho,\ i,\ r) \in \mathcal{R}_{\mathsf{wr}}$. By Theorem 5.6 we get $\mathcal{R}_{\mathsf{wr}} \vdash_\mu j_k$, for all $k < i$, hence applying $\mathsf{wrong}(\rho,\ i,\ r)$, we get $\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow \mathsf{wrong}$.

Case: $\mathsf{pev}_?(\rho, i, ?)$ Suppose $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r')$ and $i \in 1..n$, by hypothesis, for all $k < i$, $\tau_{|_k}$ is a complete partial evaluation tree of j_k, hence we know that $\mathcal{R} \vdash_\mu j_k$ holds. Set $c_i = C(\rho, i)$, then, since τ is stuck, by definition of $\longrightarrow_{\mathcal{R}}$ (cf. Fig. 3 clause (TR-5)), the subtree $\tau_{|_i}$ is stuck as well and $\mathsf{r}(\tau_{|_i}) = c_i \Rightarrow ?$. By Theorem 5.6, we get $\mathcal{R}_{\mathsf{wr}} \vdash_\mu j_k$, for all $k < i$, and, by induction hypothesis, we get $\mathcal{R}_{\mathsf{wr}} \vdash_\mu c_i \Rightarrow \mathsf{wrong}$, hence, applying rule $\mathsf{prop}(\rho, i, \mathsf{wrong})$, we get $\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow \mathsf{wrong}$. □

Lemma 5.11. *If $\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow$ wrong, then there is a stuck partial evaluation tree τ with $\mathsf{r}(\tau) = c \Rightarrow ?$.*

Proof. The proof is by induction on rules in $\mathcal{R}_{\mathsf{wr}}$. It is enough to consider only rules with wrong in the conclusion, hence we have the following three cases:

Case: $\mathit{wrong}(c)$ By Definition 5.5, there is no rule $\rho \in \mathcal{R}$ such that $C(\rho) = c$, thus $\dfrac{}{c \Rightarrow ?}$ is stuck.

Case: $\mathit{wrong}(\rho,\ i,\ r)$ By Definition 5.5, assuming $\rho \equiv \mathsf{rule}(j_1 \dots j_n,\ c,\ r')$, there is no rule $\rho' \sim_i \rho$ such that $R(\rho', i) = r$; then, by Theorem 5.6, for all $k \leq i$, $\mathcal{R} \vdash_\mu j_k$ holds, hence there is a finite and complete partial evaluation tree τ_k with $\mathsf{r}(\tau_k) = j_k$. Therefore, applying rule $\mathsf{pev}_?(\rho, i, r)$ to $\tau_1, \dots, \tau_i$, we get a partial evaluation tree, which is stuck, by definition of $\longrightarrow_{\mathcal{R}}$.

Case: $\mathsf{prop}(\rho, i, \mathit{wrong})$ Suppose $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ and $c_i = C(j_i)$, then, by induction hypothesis, we get that there is a stuck tree τ' such that $\mathsf{r}(\tau') = c_i \Rightarrow ?$; then, by Theorem 5.6, for all $k < i$, $\mathcal{R} \vdash_\mu j_k$ holds, hence there is a finite and complete partial evaluation tree τ_k with $\mathsf{r}(\tau_k) = j_k$. Therefore, applying $\mathsf{pev}_?(\rho, i, ?)$ to $\tau_1, \dots, \tau_{i-1}, \tau'$, we get a stuck tree. □

Theorem 5.12. *$\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow$ wrong iff $\dfrac{}{c \Rightarrow ?} \longrightarrow^\star_{\mathcal{R}} \tau$, where τ is stuck.*

Proof. $\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow \mathsf{wrong}$ implies $\dfrac{}{c \Rightarrow ?} \longrightarrow^\star_{\mathcal{R}} \tau$ where τ is stuck. By Lemma 5.11 we get a stuck partial evaluation tree τ with $\mathsf{r}(\tau) = c \Rightarrow ?$, hence the thesis follows by Proposition 4.13 (2), as we trivially have $\dfrac{}{c \Rightarrow ?} \sqsubseteq \tau$.

$\dfrac{}{c \Rightarrow ?} \longrightarrow^\star_{\mathcal{R}} \tau$ where τ is stuck implies $\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow \mathsf{wrong}$. It follows immediately from Lemma 5.10, since $\mathsf{r}(\tau) = c \Rightarrow ?$ by hypothesis. □

6 DIVERGENCE BY COAXIOMS

As we have described in Section 5.1, traces allow us to explicitly model divergence, provided that we interpret rules coinductively: a configuration diverges if it evaluates to an infinite trace. However, the resulting semantics is somewhat redundant: traces keep track of all configurations visited during the evaluation, while we are just interested in whether there is a final result or non-termination, and a configuration may evaluate to many different infinite traces, hence divergence is modelled in many ways. In this section we show how coaxioms (cf. Definition 2.1 in Section 2) can be succesfully adopted to achieve a more abstract model of divergence, removing this redundancy. Basically, we present a systematic definition of the approach discussed by Ancona et al. [9].

$$\text{(DIV-APP-1)}\ \frac{e_1 \Rightarrow \infty}{e_1\ e_2 \Rightarrow \infty} \qquad \text{(DIV-APP-2)}\ \frac{e_1 \Rightarrow \lambda x.e \quad e_2 \Rightarrow \infty}{e_1\ e_2 \Rightarrow \infty}$$

$$\text{(DIV-APP-3)}\ \frac{e_1 \Rightarrow \lambda x.e \quad e_2 \Rightarrow v_2 \quad e[v_2/x] \Rightarrow \infty}{e_1\ e_2 \Rightarrow \infty}$$

Fig. 7. Divergence propagation rules for application

The key idea is to regard divergence just as a special result ∞, that, like infinite traces (cf. Definition 5.1) and wrong (cf. Section 5.2), can only be propagated by big-step rules. To this end, we define yet another construction, extending a given big-step semantics.

Let us assume a big-step semantics $\langle C, R, \mathcal{R}\rangle$. Then, the extended judgement has shape $c \Rightarrow r_\infty$ where $r_\infty \in R_\infty = R + \{\infty\}$, that is, it is either a result or divergence. To define the extended semantics, we construct, starting from $\mathcal{R}$, a new set of rules $\mathcal{R}_\infty$ as follows:

Definition 6.1 (Rules for divergence). The set of rules $\mathcal{R}_\infty$ is obtained by adding to $\mathcal{R}$ the following rules:

divergence propagation rules For each rule $\rho = \text{rule}(j_1 \dots j_n,\ c,\ r)$ in $\mathcal{R}$ and index $i \in 1..n$, define rule $\text{prop}(\rho, i, \infty)$ as

$$\frac{j_1 \quad \dots \quad j_{i-1} \quad C(j_i) \Rightarrow \infty}{c \Rightarrow \infty}$$

These additional rules propagate divergence, that is, if a configuration in the premises of a rule in $\mathcal{R}$ diverges, then the subsequent premises are ignored and the configuration in the conclusion diverges as well. This is very similar to infinite trace rules, but here we do not need to construct traces to represent divergence. Note that the triple $\langle C, R_\infty, \mathcal{R}_\infty\rangle$ is a big-step semantics according to Definition 3.1.

Now the question is: how do we interpret such rules? The standard inductive interpretation of big-step rules, as for trace semantics, is not enough in this setting, since there is no axiom introducing ∞, hence it cannot be derived by finite derivations. In other words, the inductive interpretation of $\mathcal{R}_\infty$ can only capture converging computations, hence it is equivalent to the inductive interpretation of $\mathcal{R}$. On the other hand, differently from trace semantics, even the coinductive interpretation cannot provide the expected semantics: it allows the derivation of too many judgements. For instance, in Fig. 7, we report the divergence propagation rules obtained starting from meta-rule (APP) of the example in Fig. 1 (for other meta-rules the outcome is analogous); then, using these rules (and the original ones in Fig. 1), we can build the following infinite derivation for Ω, which is correct for any $r_\infty \in R_\infty$.

$$\frac{\dfrac{}{\omega \Rightarrow \omega} \quad \dfrac{}{\omega \Rightarrow \omega} \quad \dfrac{\vdots}{\Omega = (x\ x)[\omega/x] \Rightarrow r_\infty}}{\Omega \Rightarrow r_\infty}$$

Intuitively, we would like to allow infinite derivations only to derive divergence, namely, judgments of shape $c \Rightarrow \infty$. Inference systems with corules are precisely the tool enabling this kind of refinement. That is, in addition to divergence propagation rules, we can add appropriate corules $\mathcal{R}_{\text{co}}$ for divergence, as defined below.

Definition 6.2 (Coaxioms for divergence). The set of corules $\mathcal{R}_{\text{co}}$ consists of the following coaxioms:

coaxioms for divergence for each configuration $c \in C$, define coaxiom $\text{div}_{\text{co}}(c)$ as $\dfrac{}{c \Rightarrow \infty}$.

As described in Section 2, coaxioms impose additional conditions on infinite derivations to be considered correct: a judgement $c \Rightarrow r_\infty$ is derivable in $\langle \mathcal{R}_\infty, \mathcal{R}_{\text{co}}\rangle$ iff it has an arbitrary (finite or infinite) derivation in $\mathcal{R}_\infty$, whose nodes all

have a finite derivation in $\mathcal{R}_\infty \cup \mathcal{R}_{\mathrm{co}}$, that is, using both rules and corules. We will write $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathrm{co}} \rangle \vdash_\nu c \Rightarrow r_\infty$ when $c \Rightarrow r_\infty$ is derivable in $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathrm{co}} \rangle$.

In the above example, $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathrm{co}} \rangle \vdash_\nu \Omega \Rightarrow r_\infty$ holds iff $r_\infty = \infty$, because $\Omega \Rightarrow r$ has no finite derivation in $\mathcal{R}_\infty \cup \mathcal{R}_{\mathrm{co}}$, for any $r \in R$. In the case of the trace construction (cf. Section 5.1), coaxioms are not needed as rules are *productive*, because the trace in the conclusion is always strictly larger than those in the premises, see Definition 5.1.

To check that the construction in Definition 6.1 and Definition 6.2 is a correct extension of the given big-step semantics, as for trace semantics, we have to show it is conservative, in the sense that it does not affect the semantics of converging computations, as formally stated below.

Theorem 6.3. *$\langle \mathcal{R}_\infty, \mathcal{R}_{\mathrm{co}} \rangle \vdash_\nu c \Rightarrow r$ iff $\mathcal{R} \vdash_\mu c \Rightarrow r$.*

Proof. The right-to-left implication is trivial as $\mathcal{R} \subseteq \mathcal{R}_\infty$ by Definition 6.1. To get the other direction, note that if $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathrm{co}} \rangle \vdash_\nu c \Rightarrow r$ then we have $\mathcal{R}_\infty \cup \mathcal{R}_{\mathrm{co}} \vdash_\mu c \Rightarrow r$. Hence, we prove by induction on rules in $\mathcal{R}_\infty \cup \mathcal{R}_{\mathrm{co}}$ that, if $\mathcal{R}_\infty \cup \mathcal{R}_{\mathrm{co}} \vdash_\mu c \Rightarrow r$ then $\mathcal{R} \vdash_\mu c \Rightarrow r$. The cases of coaxiom $\mathrm{div}_{\mathrm{co}}(c)$ and divergence propagation $\mathrm{prop}(\rho, i, \infty)$ are both empty, as the conclusion of such rules has shape $c \Rightarrow \infty$. The only relevant case is that of a rule $\rho \in \mathcal{R}$, for which the thesis follows immediately. □

Inference systems with corules come with the bounded coinduction principle (cf. Theorem 2.2). Thanks to such principle, we can define a coinductive proof principle, which allows us to prove that a predicate on configurations ensures the existence of a non-terminating computation.

Lemma 6.4. *Let $\mathcal{S} \subseteq C$ be a set. If, for all $c \in \mathcal{S}$, there are $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ in $\mathcal{R}$ and $i \in 1..n$ such that*

(1) for all $k < i$, $\mathcal{R} \vdash_\mu j_k$, and
(2) $C(j_i) \in \mathcal{S}$

then, for all $c \in \mathcal{S}$, $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathrm{co}} \rangle \vdash_\nu c \Rightarrow \infty$.

Proof. Consider the set $\mathcal{S}' = \{\langle c, \infty \rangle \mid c \in \mathcal{S}\} \cup \{\langle c, r \rangle \mid \mathcal{R} \vdash_\mu c \Rightarrow r\}$, then the proof is by bounded coinduction (cf. Theorem 2.2).

Boundedness We have to show that, for all $\langle c, r_\infty \rangle \in \mathcal{S}'$, $\mathcal{R}_\infty \cup \mathcal{R}_{\mathrm{co}} \vdash_\mu c \Rightarrow r_\infty$ holds. This is easy because, if $r_\infty = \infty$, then this holds by coaxiom $\mathrm{div}_{\mathrm{co}}(c)$, otherwise $r_\infty \in R$ and $\mathcal{R} \vdash_\mu c \Rightarrow r_\infty$, hence this holds since $\mathcal{R} \subseteq \mathcal{R}_\infty \subseteq \mathcal{R}_\infty \cup \mathcal{R}_{\mathrm{co}}$.

Consistency We have to show that, for all $\langle c, r_\infty \rangle \in \mathcal{S}'$, there is a rule $\langle j_1 \dots j_n, c \Rightarrow r_\infty \rangle \in \mathcal{R}_\infty$ such that, for all $k \in 1..n$, $\langle C(j_k), R_\infty(j_k) \rangle \in \mathcal{S}'$. There are two cases:

- If $r_\infty = \infty$, then by hypothesis (Item 1), we have a rule $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r) \in \mathcal{R}$ and an index $i \in 1..n$ such that, for all $k < i$, $\mathcal{R} \vdash_\mu j_k$ and $C(j_i) \in \mathcal{S}$. Then, the needed rule is $\mathrm{prop}(\rho, i, \infty)$.
- If $r_\infty \in R$, then, by construction of $\mathcal{S}'$, we have $\mathcal{R} \vdash_\mu c \Rightarrow r_\infty$, hence, there is a rule $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r_\infty) \in \mathcal{R} \subseteq \mathcal{R}_\infty$, where, for all $k \in 1..n$, $\mathcal{R} \vdash_\mu j_k$ holds, and so $\langle C(j_k), R(j_k) \rangle \in \mathcal{S}'$.

□

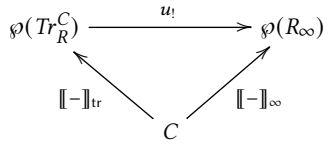
The reader may have noticed that most definitions and results in this section are very similar to those provided for trace semantics in Section 5.1. This is not a coincidence, indeed, we now formally prove this semantics is an *abstraction* of trace semantics.

Intuitively, if we are only interested in modelling convergence or divergence, traces are useless, in the sense that it is only relevant to know whether the trace is infinite or not and, in case it is finite, the final result. We can model this intuition by a (surjective) function $u : Tr_R^C \to R_\infty$ simply forgetting traces, that is, $u(\langle t, r\rangle) = r$ and $u(\sigma) = \infty$, with $t \in C^\star$ and $\sigma \in C^\omega$.

Then, we aim at proving the following result:

Theorem 6.5. *$\langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}}\rangle \vdash_\nu c \Rightarrow r_\infty$ iff $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} r_{\mathsf{tr}}$, for some r_{tr} such that $r_\infty = u(r_{\mathsf{tr}})$.*

In a diagrammatic form, Theorem 6.5 says that the following diagram commutes:



where $u_! : \wp(\mathcal{R}_{\mathsf{tr}}) \to \wp(\mathcal{R}_\infty)$ is the direct image of u, $[\![-]\!]_{\mathsf{tr}} : C \to \wp(Tr_R^C)$ is defined by $[\![c]\!]_{\mathsf{tr}} = \{r_{\mathsf{tr}} \in Tr_R^C \mid \mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} r_{\mathsf{tr}}\}$, and $[\![-]\!]_\infty : C \to \wp(R_\infty)$ is defined by $[\![c]\!]_{\mathsf{tr}} = \{r_\infty \in R_\infty \mid \langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}}\rangle \vdash_\nu c \Rightarrow r_\infty\}$.

Proof. The statement can be split in the following two points:

(1) $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}}\rangle \vdash_\nu c \Rightarrow r$ iff $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$, for some $t \in C^\star$, and

(2) $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}}\rangle \vdash_\nu c \Rightarrow \infty$ iff $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \sigma$, for some $\sigma \in C^\omega$.

The first point follows immediately from Theorem 5.3 and Theorem 6.3, as $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}}\rangle \vdash_\nu c \Rightarrow r$ and $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \langle t, r\rangle$ are both equivalent to $\mathcal{R} \vdash_\mu c \Rightarrow r$. Then, we have only to prove the second point.

The left-to-right implication follows applying Lemma 5.4 to the set $\mathcal{S}_\infty = \{c \in C \mid \langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}}\rangle \vdash_\nu c \Rightarrow \infty\}$. If $c \in \mathcal{S}_\infty$, then $c \Rightarrow \infty$ is derived by a rule $\mathsf{prop}(\rho, i, \infty)$ for some $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ in $\mathcal{R}$ and $i \in 1..n$, hence we have $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}}\rangle \vdash_\nu j_k$, which implies $\mathcal{R} \vdash_\mu j_k$ by Theorem 6.3, for all $k < i$, and $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}}\rangle \vdash_\nu C(j_i) \Rightarrow \infty$, that is, $C(j_i) \in \mathcal{S}_\infty$, because these judgements are the premises of $\mathsf{prop}(\rho, i, \infty)$. Therefore, the hypotheses of Lemma 5.4 are satisfied and we get, for all $c \in \mathcal{S}_\infty$, $\mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \sigma_c$, for some $\sigma_c \in C^\omega$, hence $u(\sigma_c) = \infty$.

Similarly, the right-to-left implication follows applying Lemma 6.4 to the set $\mathcal{S}_{\mathsf{tr}} = \{c \in C \mid \mathcal{R}_{\mathsf{tr}} \vdash_\nu c \Rightarrow_{\mathsf{tr}} \sigma \text{ for some } \sigma \in C^\omega\}$. If $c \in \mathcal{S}_{\mathsf{tr}}$, then, for some $\sigma \in C^\omega$, $c \Rightarrow_{\mathsf{tr}} \sigma$ is derived by a rule $\mathsf{trace}_\infty(\rho,\ i,\ t_1, \dots, t_{i-1},\ \sigma')$, for some $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ in $\mathcal{R}$ and $i \in 1..n$, hence we have $\mathcal{R}_{\mathsf{tr}} \vdash_\nu C(j_k) \Rightarrow_{\mathsf{tr}} \langle t_k, R(j_k)\rangle$, which implies $\mathcal{R} \vdash_\mu j_k$ by Theorem 5.3, for all $k < i$, and $\mathcal{R}_{\mathsf{tr}} \vdash_\nu C(j_i) \Rightarrow_{\mathsf{tr}} \sigma'$, that is, $C(j_i) \in \mathcal{S}_{\mathsf{tr}}$, because these judgements are the premises of the rule $\mathsf{trace}_\infty(\rho,\ i,\ t_1, \dots, t_{i-1},\ \sigma')$. Therefore, the hypotheses of Lemma 6.4 are satisfied and we get, for all $c \in \mathcal{S}_{\mathsf{tr}}$, $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}}\rangle \vdash_\nu c \Rightarrow \infty$. □

As an immediate consequence of Theorem 6.5 and Theorem 5.7, we get the following corollary, stating that the construction given by Definitions 6.1 and 6.2 correctly models diverging computations:

Corollary 6.6. *$\langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}}\rangle \vdash_\nu c \Rightarrow \infty$ iff $\dfrac{}{c \Rightarrow ?} \longrightarrow_{\mathcal{R}}^{\omega}$.*

Total semantics. We now briefly describe how we can combine the presented constructions in order to get a semantics modelling *all* computations as defined in Section 4.2. In particular, we will use the wrong construction to model stuck computations and the construction in this section to model divergence, because they are more similar to each other.

Let us consider a big-step semantics $\langle C, R, \mathcal{R}\rangle$. We add to R two special values to model stuckness and divergence, defining $R_{\mathsf{tot}} = R + \{\mathsf{wrong}\} + \{\infty\}$. Then, we have to add appropriate rules to handle these two special results: the idea

is to add "simultanously" rules from Definition 5.5 and from Definition 6.1, that is, we define $\mathcal{R}_{\mathsf{tot}} = \mathcal{R}_{\mathsf{wr}} \cup \mathcal{R}_{\infty}$. Note that, since both $\mathcal{R}_{\mathsf{wr}}$ and $\mathcal{R}_{\infty}$ extend $\mathcal{R}$, we have $\mathcal{R} \subseteq \mathcal{R}_{\mathsf{tot}}$. In addition, the triple $\langle C, R_{\mathsf{tot}}, \mathcal{R}_{\mathsf{tot}} \rangle$ is a big-step semantics according to Definition 3.1. Finally, to properly model divergence, we have to add corules from Definition 6.2, so that infinite derivations are only allowed to prove divergence.

Since, as we have noticed, all the presented constructions yield a big-step semantics, starting from another one, we can also try to combine them "sequentially". Of course, there are two possibilities: either we first apply the wrong construction or the divergence construction. Nicely, it is not difficult to check that all these possibilities yield the same big-step semantics $\langle C, R_{\mathsf{tot}}, \mathcal{R}_{\mathsf{tot}} \rangle$, as depicted below:

$$\begin{array}{ccc} \langle C, R, \mathcal{R} \rangle & \xrightarrow{\mathsf{wr}} & \langle C, R_{\mathsf{wr}}, \mathcal{R}_{\mathsf{wr}} \rangle \\ {\scriptstyle\infty}\downarrow & \searrow^{\mathsf{tot}} & \downarrow{\scriptstyle\infty} \\ \langle C, R_{\infty}, \mathcal{R}_{\infty} \rangle & \xrightarrow[\mathsf{wr}]{} & \langle C, R_{\mathsf{tot}}, \mathcal{R}_{\mathsf{tot}} \rangle \end{array}$$

Thanks to the commutativity of the above diagram, we can exploit results proved for the various constructions to get properties of this last construction, as stated below.

Proposition 6.7. *The following facts hold:*

(1) $\langle \mathcal{R}_{\mathsf{tot}}, \mathcal{R}_{\mathsf{co}} \rangle \vdash_{\nu} c \Rightarrow r$ *iff* $\mathcal{R} \vdash_{\mu} c \Rightarrow r$,

(2) $\langle \mathcal{R}_{\mathsf{tot}}, \mathcal{R}_{\mathsf{co}} \rangle \vdash_{\nu} c \Rightarrow \mathit{wrong}$ *iff* $\mathcal{R}_{\mathsf{wr}} \vdash_{\mu} c \Rightarrow \mathit{wrong}$,

(3) $\langle \mathcal{R}_{\mathsf{tot}}, \mathcal{R}_{\mathsf{co}} \rangle \vdash_{\nu} c \Rightarrow \infty$ *iff* $\langle \mathcal{R}_{\infty}, \mathcal{R}_{\mathsf{co}} \rangle \vdash_{\nu} c \Rightarrow \infty$.

Proof. All right-to-left implication are trivial, as $\mathcal{R}, \mathcal{R}_{\mathsf{wr}}, \mathcal{R}_{\infty} \subseteq \mathcal{R}_{\mathsf{tot}}$. The other implications follow from Theorems 5.6 and 6.3, relying on the above commutative diagram. □

Corollary 6.8. *For any configuration* $c \in C$*, one of the following holds:*

- *either* $\langle \mathcal{R}_{\mathsf{tot}}, \mathcal{R}_{\mathsf{co}} \rangle \vdash_{\nu} c \Rightarrow r$*, for some* $r \in R$,
- *or* $\langle \mathcal{R}_{\mathsf{tot}}, \mathcal{R}_{\mathsf{co}} \rangle \vdash_{\nu} c \Rightarrow \infty$,
- *or* $\langle \mathcal{R}_{\mathsf{tot}}, \mathcal{R}_{\mathsf{co}} \rangle \vdash_{\nu} c \Rightarrow \mathit{wrong}$.

Proof. Straightforward from Proposition 6.7 and Theorems 5.7, 5.12 and 6.5, since the partial evaluation tree $\dfrac{}{c \Rightarrow ?}$, either converges to a tree, which is either complete or stuck, or diverges. □

Note that these three possibilities in general are not mutually exclusive, that is, for instance, a configuration can both converge to a result and diverge. This is due to the fact that big-step rules can define a non-deterministic behaviour.

7 EXPRESSING AND PROVING SOUNDNESS

A predicate (for instance, a typing judgment) is *sound* when, informally, a program satisfying such predicate (*e.g.*, a well-typed program) cannot *go wrong*, following Robin Milner's slogan [39]. In small-step style, as firstly formulated by Wright and Felleisen [53], this is naturally expressed as follows: well-typed programs never reduce to terms which neither are values, nor can be further reduced (called *stuck* terms). The standard technique to ensure soundness is by subject reduction (well-typedness is preserved by reduction) and progress (a well-typed term is not stuck).

In standard (inductive) big-step semantics, soundness, as described above, cannot even be expressed, because diverging and stuck computations are not distinguishable.

Constructions presented in the previous sections make this distinction explicit, hence they allow us to reason about soundness with respect to a big-step semantics. In this section, we discuss how soundness can be expressed and we will provide sufficient conditions. In other words, we provide a proof technique to show the soundness of a predicate with respect to a big-step semantics.

It is important to highlight the following about the presented approach to soundness. First, even though type systems are the paradigmatic example, we will consider a generic predicate on configurations, hence our approach could be instantiated with other kinds of predicates. Second, depending on the kind of construction considered, we can express different flavours of soundness, which will have different proof techniques. Finally, and more importantly, as mentioned in the introduction, the extended semantics is only needed to prove the correctness of the technique, whereas to *apply* the technique for a given big-step semantics it is enough to reason on the original rules.

7.1 Expressing soundness

In the following, we assume a big-step semantics $\langle C, R, \mathcal{R} \rangle$, and an *indexed predicate on configurations and results*, that is, a family $\Pi = \langle \Pi_\iota^C, \Pi_\iota^R \rangle_{\iota \in I}$, for I set of *indexes*, with $\Pi_\iota^C \subseteq C$ and $\Pi_\iota^R \subseteq R$. A representative case is that, as in the examples of Section 8, predicates on configurations and results are typing judgments and the indexes are types; however, this setting is more general and so the proof technique could be applied to other kinds of predicates. When there is no ambiguity, we also denote by Π^C and Π^R, respectively, the corresponding predicates $\bigcup_{\iota \in I} \Pi_\iota^C$ and $\bigcup_{\iota \in I} \Pi_\iota^R$ on C and R (*e.g.*, to be well-typed with an arbitrary type).

To discuss how to express soundness of Π, first of all note that, in the non-deterministic case (that is, there is possibly more than one computation for a configuration), we can distinguish two flavours of soundness, see, *e.g.*, [29]:

soundness-must (or simply soundness) no computation can be stuck
soundness-may at least one computation is not stuck

Soundness-must is the standard soundness in small-step semantics, and can be expressed by the wrong construction as follows:

soundness-must If $c \in \Pi^C$, then $\mathcal{R}_{\mathsf{wr}} \not\vdash_\mu c \Rightarrow \mathsf{wrong}$

Soundness-must *cannot* be expressed by the constructions making divergence explicit, because stuck computations are not explicitly modelled. In contrast, soundness-may can be expressed, for instance, by the divergence construction as follows:

soundness-may If $c \in \Pi^C$, then $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathsf{co}} \rangle \vdash_\nu c \Rightarrow r_\infty$, for some $r_\infty \in R_\infty$

whereas it cannot be expressed by the wrong construction, since diverging computations are not modelled. Note that, instead, using the total semantics, we can express both flavours of soundness, as it models both diverging and stuck computations.

Of course soundness-must and soundness-may coincide in the deterministic case. Finally, note that indexes (*e.g.*, the specific types of configurations and results) do not play any role in the above statements. However, they are relevant in the notion of *strong soundness*, introduced by Wright and Felleisen [53]. Strong soundness holds (in must or may flavour) if soundness holds (in must or may flavour), and, moreover, configurations satisfying Π_ι^C (*e.g.*, having a given type) produce results, if any, satisfying Π_ι^R (*e.g.*, of the same type). Note that soundness alone does not even guarantee to obtain a result satisfying Π^R (*e.g.*, a well-typed result). The sufficient conditions introduced in the following subsection actually ensure strong soundness.

In Section 7.2, we provide sufficient conditions for soundness-must, showing that they ensure soundness as stated above (Theorem 7.6). Then, in Section 7.3, we provide (weaker) sufficient conditions for soundness-may, and show that they ensure soundness-may (Theorem 7.9).

7.2 Conditions ensuring soundness-must

The three conditions which ensure the soundness-must property are *local preservation*, $\exists$*-progress*, and $\forall$*-progress*. The names suggest that the former plays the role of the *type preservation (subject reduction)* property, and the latter two of the *progress* property in small-step semantics. However, as we will see, the correspondence is only rough, since the reasoning here is different.

Considering the first condition more closely, we use the name *preservation* rather than type preservation since, as already mentioned, the proof technique can be applied to arbitrary predicates. More importantly, *local* means that the condition is *on single rules* rather than on the semantic relation as a whole, as standard subject reduction; the semantic relation is only used in the hypotheses of the condition, so that, when checking it, one can rely on stronger assumptions. The same holds for the other two conditions.

Definition 7.1 (Local preservation (LP)). For each $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ in $\mathcal{R}$, if $c \in \Pi^C_\iota$, then there exists $\iota_1, \dots, \iota_n \in I$ such that

(1) for all $k \in 1..n$, if, for all $h < k$, $\mathcal{R} \vdash_\mu j_h$ and $R(j_h) \in \Pi^R_{\iota_h}$, then $C(j_k) \in \Pi^C_{\iota_k}$, and
(2) if, for all $k \in 1..n$, $\mathcal{R} \vdash_\mu j_k$ and $R(j_k) \in \Pi^R_{\iota_k}$, then $r \in \Pi^R_\iota$.

Thinking to the paradigmatic case where the indexes are types, to check that this condition holds, for each rule $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ where c, the conclusion, has type ι, we have to find types $\iota_1, \dots, \iota_n$, which can be assigned to (configurations and results in) the premises, and, when all the premises satisfy the chosen type, r, the result in the conclusion, must have type ι, that is, the same type of c. More precisely, we will proceed as follows: we start finding type ι_1, and successively find the type ι_k for (the configuration in) the k-th premise assuming that all previous premises are derivable and their results have the expected types, and, finally, we have to check that the final result r has type ι assuming all premises are derivable and their results have the expected type. Indeed, if all such previous premises are derivable, then the expected type should be preserved by their results; if some premise is not derivable, the considered rule is "useless". For instance, considering (an instantiation of) meta-rule (APP) $\mathsf{rule}(e_1 \Rightarrow \lambda x.e\ \ e_2 \Rightarrow v_2\ \ e[v_2/x] \Rightarrow v,\ e_1\ e_2,\ v)$ in Fig. 1, we prove that $e[v_2/x]$ has the type T of $e_1\ e_2$ under the assumption that $\lambda x.e$ has type $T' \to T$, and v_2 has type T' (see the proof example in Section 8.1 for more details). A counter-example to condition (LP) is discussed at the beginning of Section 8.3.

The following lemma states that local preservation actually implies *preservation* of the semantic relation as a whole.

LEMMA 7.2 (PRESERVATION). *Let* $\langle C, R, \mathcal{R} \rangle$ *and* $\Pi = \langle \Pi^C_\iota, \Pi^R_\iota \rangle_{\iota \in I}$ *satisfy condition (LP). If* $\mathcal{R} \vdash_\mu c \Rightarrow r$ *and* $c \in \Pi^C_\iota$, *then* $r \in \Pi^R_\iota$.

PROOF. The proof is by a double induction From the hypotheses, we know that $c \Rightarrow r$ has a finite derivation in $\mathcal{R}$ and $c \in \Pi^C_\iota$. The first induction is on the derivation of $c \Rightarrow r$. Suppose the last applied rule is $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ and denote by *RH* the induction hypothesis. Then, we prove by complete arithmetic induction on $k \in 1..n$ (the second induction) that $C(j_k) \in \Pi_{\iota_k}$, for all $k \in 1..n$ and for some $\iota_1, \dots, \iota_n \in I$. Let us denote by *IH* the second induction hypothesis. By (LP), there are indexes $\iota_1, \dots, \iota_n \in I$, satisfying Items 1 and 2 of (LP) (cf. Definition 7.1). Let $k \in 1..n$, then

by *IH* we know that $C(j_h) \in \Pi^C_{\iota_h}$, for all $h < k$. Then, by *RH*, we get that $R(j_h) \in \Pi^R_{\iota_h}$. Hence, by (LP) (cf. Definition 7.1 (1)), we get $C(j_k) \in \Pi_{\iota_k}$, as needed.

Now, since $C(j_k) \in \Pi^C_{\iota_k}$, for all $k \in 1..n$, as we have just proved, again by *RH*, we get that $R(j_k) \in \Pi^R_{\iota_k}$, for all $k \in 1..n$. Then, by (LP) (cf. Definition 7.1 (2)), we conclude that $r \in \Pi^R_\iota$, as needed. □

The following proposition is a form of local preservation where indexes (*e.g.*, specific types) are not relevant, simpler to use in the proofs of Theorems 7.6 and 7.9.

Proposition 7.3. *Let* $\langle C, R, \mathcal{R}\rangle$ *and* $\Pi = \langle \Pi^C_\iota, \Pi^R_\iota\rangle_{\iota\in I}$ *satisfy condition (LP). For each rule* $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ *and* $k \in 1..n$*, if* $c \in \Pi^C$ *and, for all* $h < k$*,* $\mathcal{R} \vdash_\mu j_h$*, then* $C(j_k) \in \Pi^C$*.*

Proof. By hypothesis we know that $c \in \Pi^C_\iota$, for some $\iota \in I$, thus by condition (LP), there are indexes $\iota_1, \dots, \iota_n \in I$, satisfying Items 1 and 2 of (LP) (cf. Definition 7.1). We show by complete arithmetic induction that, for all $k \in 1..n$, $C(j_k) \in \Pi^C_{\iota_k}$, which implies the thesis. Assume the thesis for all $h < k$, then, since by hypothesis we have $\mathcal{R} \vdash_\mu j_h$ for all $h < k$, we get, by induction hypothesis, $C(j_h) \in \Pi^C_{\iota_h}$, for all $h < k$. By Lemma 7.2, we also get $R(j_h) \in \Pi^R_{\iota_h}$, hence, by condition (LP) (cf. Definition 7.1 (1)), we get $C(j_k) \in \Pi^C_{\iota_k}$, as needed. □

The second condition, named ∃*-progress*, ensures that, for configurations satisfying Π (*e.g.*, well-typed), we can *start* the evaluation, that is, the construction of an evaluation tree.

Definition 7.4 (∃-progress (∃P)). For each $c \in \Pi^C$, there exists a rule $\rho \in \mathcal{R}$ such that $C(\rho) = c$.

The third condition, named ∀*-progress*, ensures that, for configurations satisfying Π (*e.g.*, well-typed), we can *continue* the evaluation, that is, the construction of the evaluation tree. This condition uses the equivalence on rules introduced in Definition 4.12.

Definition 7.5 (∀-progress (∀P)). For each rule $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$ with $c \in \Pi^C$, for each $k \in 1..n$, if, for all $h < k$, $\mathcal{R} \vdash_\mu j_h$ and $\mathcal{R} \vdash_\mu C(j_k) \Rightarrow r'$, for some $r' \in R$, then there is a rule $\rho' \sim_k \rho$ such that $R(\rho', k) = r'$.

We have to check, for each rule $\rho = \mathsf{rule}(j_1 \dots j_n,\ c,\ r)$, the following: if the configuration c in the conclusion satisfies the predicate (*e.g.*, is well-typed), then, for each $k \in 1..n$, if the configuration in the k-th premise evaluates to some result r' (that is, $\mathcal{R} \vdash_\mu C(j_k) \Rightarrow r'$), then there is a rule (ρ itself or another rule with the same configuration in the conclusion and the same first $k-1$ premises) with such judgement as k-th premise. This check can be done under the assumption that all the previous premises are derivable. For instance, consider again (an instantiation of) the meta-rule (APP) $\mathsf{rule}(e_1 \Rightarrow \lambda x.e\ \ e_2 \Rightarrow v_2\ \ e[v_2/x] \Rightarrow v,\ e_1\ e_2,\ v)$. Assuming that e_1 evaluates to some v_1, we have to check that there is a rule with first premise $e_1 \Rightarrow v_1$, in practice, that v_1 is a λ-abstraction; in general, checking (∀P) for a (meta-)rule amounts to show that configurations in the premises evaluate to results with the required shape (see also the proof example in Section 8.1).

We now prove the claim of soundness-must expressed by means of the wrong construction (cf. Section 5.2).

Theorem 7.6 (Soundness-must). *Let* $\langle C, R, \mathcal{R}\rangle$ *and* $\Pi = \langle \Pi^C_\iota, \Pi^R_\iota\rangle_{\iota\in I}$ *satisfy conditions (LP), (∃P) and (∀P). If* $c \in \Pi^C$*, then* $\mathcal{R}_{\mathsf{wr}} \nvdash_\mu c \Rightarrow$ *wrong.*

Proof. To prove the statement, we assume $\mathcal{R}_{\mathsf{wr}} \vdash_\mu c \Rightarrow \mathsf{wrong}$ and look for a contradiction. The proof is by induction on the derivation of $c \Rightarrow \mathsf{wrong}$. We split cases on the last applied rule in such derivation.

Case: $\mathit{wrong}(c)$ By construction (cf. Definition 5.5), we know that there is no rule $\rho \in \mathcal{R}$ such that $C(\rho) = c$, and this violates condition ($\exists$P), since $c \in \Pi^C$, by hypothesis.

Case: $\mathit{wrong}(\rho, i, r')$ Suppose $\rho = \mathsf{rule}(j_1 \dots j_n, c, r)$, hence $i \in 1..n$, then, by hypothesis, for all $k < i$, we have $\mathcal{R}_{\mathsf{wr}} \vdash_\mu j_k$, and $\mathcal{R}_{\mathsf{wr}} \vdash_\mu C(j_i) \Rightarrow r'$, and these judgments can also be derived in $\mathcal{R}$ by conservativity (cf. Theorem 5.6). Furthermore, by construction (cf. Definition 5.5), we know that there is no other rule $\rho' \sim_i \rho$ such that $R(\rho', i) = r'$, and this violates condition ($\forall$P), since $c \in \Pi^C$ by hypothesis.

Case: $\mathsf{prop}(\rho, i, \mathit{wrong})$ Suppose $\rho = \mathsf{rule}(j_1 \dots j_n, c, r)$, hence $i \in 1..n$, then, by hypothesis, for all $k < i$, we have $\mathcal{R}_{\mathsf{wr}} \vdash_\mu j_k$, and these judgments can also be derived in $\mathcal{R}$ by conservativity (cf. Theorem 5.6). Then, by Proposition 7.3 (which requires condition (LP)), since $c \in \Pi^C$, we have $C(j_i) \in \Pi^C$, hence we get the thesis by induction hypothesis, because $\mathcal{R}_{\mathsf{wr}} \vdash_\mu C(j_i) \Rightarrow \mathsf{wrong}$ holds by hypothesis.

□

Note that conditions (LP), ($\exists$P) and ($\forall$P), actually ensure strong soundness, because, by Lemma 7.2, which is applicable since we assume (LP), we have that converging computations preserve indexes of the predicate.

7.3 Conditions ensuring soundness-may

As discussed in Section 7.1, if we explicitly model divergence rather than stuck computations, we can only express a weaker form of soundness: at least one computation is not stuck (*soundness-may*). Actually, we will state soundness-may in a different, but equivalent, way, which is simpler to prove, that is, a configuration that does not converge, diverges.

As the reader can expect, to ensure this property weaker sufficient conditions are enough: namely, condition (LP), and another condition, named *may-progress*, defined below. We write "$\mathcal{R} \not\vdash_\mu c \Rightarrow$ " if *c does not converge* (there is no r such that $\mathcal{R} \vdash_\mu c \Rightarrow r$).

Definition 7.7 (May-progress (MAYP)). For each $c \in \Pi^C$, there is a rule $\rho = \mathsf{rule}(j_1 \dots j_n, c, r)$ such that, if there is a (first) $k \in 1..n$ such that $\mathcal{R} \not\vdash_\mu j_k$ and, for all $h < k$, $\mathcal{R} \vdash_\mu j_h$, then $\mathcal{R} \not\vdash_\mu C(j_k) \Rightarrow$.

This condition can be informally understood as follows: we have to show that there is an either finite or infinite computation for c. If we find a rule where all premises are derivable (there is no k), then there is a finite computation. Otherwise, c cannot converge. In this case, we should find a rule where the configuration in the first non-derivable premise k cannot converge as well. Indeed, by coinductive reasoning (cf. Theorem 7.9), this implies that c diverges. The following proposition states that this condition is indeed a weakening of ($\exists$P) and ($\forall$P).

Proposition 7.8. *Conditions ($\exists$P) and ($\forall$P) imply condition (MAYP).*

Proof. For each $c \in C$, let us define $b_c \in \mathbb{N}$ as $\max\{\#\rho \mid C(\rho) = c\}$, which is well-defined and finite by condition (BP) in Definition 3.1. For each rule ρ with $C(\rho) = c$, let us denote by $nd(\rho)$ the index of the first premise of ρ which is not derivable, if any, otherwise set $nd(\rho) = b_c + 1$. For each $c \in \Pi^C$, we first prove the following fact: ($\star$) for each rule ρ, with $C(\rho) = c$, there exists a rule ρ' such that $C(\rho') = c$, $nd(\rho') \geq nd(\rho)$ and, if $nd(\rho') \leq b_c$, then $\mathcal{R} \not\vdash_\mu C(\rho', nd(\rho')) \Rightarrow$. Note that the requirement in ($\star$) is the same as that of condition (MAYP). The proof is by complete arithmetic induction on $h(\rho) = b_c + 1 - nd(\rho)$. If $h(\rho) = 0$, hence $nd(\rho) = b_c + 1$, then the thesis follows by taking $\rho' = \rho$. Otherwise, we have two cases: if there is no $r \in R$ such that $\mathcal{R} \vdash_\mu C(\rho, nd(\rho)) \Rightarrow r$, then we have the thesis taking $\rho' = \rho$; otherwise, by condition ($\forall$P), there is a rule $\rho'' \sim_{nd(\rho)} \rho$ such that $R(\rho'', nd(\rho)) = r$, hence $nd(\rho'') > nd(\rho)$. Then, we have $h(\rho'') < h(\rho)$, hence we get the thesis by induction hypothesis.

Now, by ($\exists$P), there is a rule ρ with $C(\rho) = c$, and applying ($\star$) to ρ we get (MAYP). □

We now prove the claim of soundness-may expressed by means of the divergence construction (cf. Section 6).

Theorem 7.9 (Soundness-may). *Let* $\langle C, R, \mathcal{R}\rangle$ *and* $\Pi = \langle \Pi_\iota^C, R_\iota\rangle_{\iota\in I}$ *satisfy conditions (lp) and (mayp). If* $c \in \Pi^C$, *then* $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathrm{co}}\rangle \vdash_\nu c \Rightarrow r_\infty$, *for some* $r_\infty \in R_\infty$.

Proof. First note that, thanks to Theorem 6.3, the statement is equivalent to the following:

If $c \in \Pi^C$ and $\mathcal{R} \not\vdash_\mu c \Rightarrow$, then $\langle \mathcal{R}_\infty, \mathcal{R}_{\mathrm{co}}\rangle \vdash_\nu c \Rightarrow \infty$.

Then, the thesis follows by Lemma 6.4. We set $\mathcal{S} = \{c \in C \mid c \in \Pi^C \text{ and } \mathcal{R} \not\vdash_\mu c \Rightarrow \}$, and show that, for all $c \in \mathcal{S}$, there are $\rho = \mathrm{rule}(j_1 \dots j_n,\ c,\ r)$ and $k \in 1..n$ such that, for all $h < k$, $\mathcal{R} \vdash_\mu j_h$ and $C(j_k) \in \mathcal{S}$.

Consider $c \in \mathcal{S}$, then, by (mayp) (cf. Definition 7.7), there is $\rho = \mathrm{rule}(j_1 \dots j_n,\ c,\ r)$. By definition of $\mathcal{S}$, we have $\mathcal{R} \not\vdash_\mu c \Rightarrow$, hence there exists a (first) $k \in 1..n+1$ such that $\mathcal{R} \not\vdash_\mu j_k$, since, otherwise, we would have $\mathcal{R} \vdash_\mu c \Rightarrow r$. Then, since k is the first index with such property, for all $h < k$, we have $\mathcal{R} \vdash_\mu j_h$, hence, again by condition (mayp) (cf. Definition 7.7), we have that $\mathcal{R} \not\vdash_\mu C(j_k) \Rightarrow$. Finally, since $c \in \Pi^C$ and, for all $h < k$, we have $\mathcal{R} \vdash_\mu j_h$, by Proposition 7.3 we get $C(j_k) \in \Pi^C$, hence $C(j_k) \in \mathcal{S}$, as needed. □

Note that conditions (lp) and (mayp) actually ensure strong soundness, because, by Lemma 7.2, which is applicable since we assume (lp), we have that converging computations preserve indexes of the predicate.

8 EXAMPLES OF SOUNDNESS PROOFS

In this section, we show how to use the technique introduced in Section 7 to prove soundness of a type system with respect to a big-step semantics, by several examples. We focus on the technique for soundness-must, as it is the usual notion of soundness for type systems. Section 8.1 explains in detail how a typical soundness proof can be rephrased in terms of our technique, by reasoning directly on big-step rules. Section 8.2 shows a case where this is advantageous, since the property to be checked is *not preserved* by intermediate computation steps, whereas it holds for the whole computation. Section 8.3 considers a more sophisticated type system, with intersection and union types. Section 8.4 shows another example where types are not preserved, whereas soundness can be proved with our technique. This example is intended as a preliminary step towards a more challenging case. In Section 8.5 we show how our technique can also deal with imperative features.

8.1 Simply-typed λ-calculus with recursive types

As a first example, we take the λ-calculus with natural constants, successor, and non-deterministic choice introduced in Fig. 1. We consider a standard simply-typed version with (equi)recursive types, obtained by interpreting the production in the top section of Fig. 8 coinductively. Introducing recursive types makes the calculus non-normalising and permits to write interesting programs such as Ω (see Section 5.1).

The typing rules are recalled in the bottom section of Fig. 8 and, as usual, they are interpreted inductively. Type environments, written Γ, are finite maps from variables to types, and $\Gamma\{T/x\}$ denotes the map which returns T on x and coincides with Γ elsewhere. We write $\vdash e : T$ for $\emptyset \vdash e : T$.

Let $\langle C_1, R_1, \mathcal{R}_1\rangle$ be the big-step semantics described in Fig. 1 (C_1 is the set of expressions and R_1 is the set of values), and let $\Pi 1_T^C = \{e \in C_1 \mid \vdash e : T\}$ and $\Pi 1_T^R = \{v \in R_1 \mid \vdash v : T\}$, where T is a type, defined in Fig. 8, that is, $\Pi 1_T^C$ and $\Pi 1_T^R$ are the sets of expressions and values of type T, respectively. To prove the three conditions (lp), ($\exists$p) and ($\forall$p) of Section 7.2, we need lemmas of inversion, substitution and canonical forms, as in the standard technique for small-step semantics.

$$T \quad ::= \quad \mathtt{Nat} \mid T_1 \to T_2 \quad \text{types}$$

$$\text{(T-VAR)}\ \frac{}{\Gamma \vdash x : T}\ \Gamma(x) = T \qquad \text{(T-CONST)}\ \frac{}{\Gamma \vdash n : \mathtt{Nat}}$$

$$\text{(T-ABS)}\ \frac{\Gamma\{T'/x\} \vdash e : T}{\Gamma \vdash \lambda x.e : T' \to T} \qquad \text{(T-APP)}\ \frac{\Gamma \vdash e_1 : T' \to T \quad \Gamma \vdash e_2 : T'}{\Gamma \vdash e_1\ e_2 : T}$$

$$\text{(T-SUCC)}\ \frac{\Gamma \vdash e : \mathtt{Nat}}{\Gamma \vdash \mathtt{succ}\ e : \mathtt{Nat}} \qquad \text{(T-CHOICE)}\ \frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash e_1 \oplus e_2 : T}$$

Fig. 8. λ-calculus: type system

Lemma 8.1 (Inversion). *The following hold:*

(1) If $\Gamma \vdash x : T$, *then* $\Gamma(x) = T$.
(2) If $\Gamma \vdash n : T$, *then* $T = \mathtt{Nat}$.
(3) If $\Gamma \vdash \lambda x.e : T$, *then* $T = T_1 \to T_2$ *and* $\Gamma\{T_1/x\} \vdash e : T_2$.
(4) If $\Gamma \vdash e_1\ e_2 : T$, *then* $\Gamma \vdash e_1 : T' \to T$ *and* $\Gamma \vdash e_2 : T'$.
(5) If $\Gamma \vdash \mathtt{succ}\ e : T$, *then* $T = \mathtt{Nat}$ *and* $\Gamma \vdash e : \mathtt{Nat}$.
(6) If $\Gamma \vdash e_1 \oplus e_2 : T$, *then* $\Gamma \vdash e_i : T$ *with* $i \in 1, 2$.

Lemma 8.2 (Substitution). *If* $\Gamma\{T'/x\} \vdash e : T$ *and* $\Gamma \vdash e' : T'$, *then* $\Gamma \vdash e[e'/x] : T$.

Lemma 8.3 (Canonical Forms). *The following hold:*

(1) If $\vdash v : T' \to T$, *then* $v = \lambda x.e$.
(2) If $\vdash v : \mathtt{Nat}$, *then* $v = n$.

Theorem 8.4 (Soundness). *The big-step semantics* $\langle C_1, R_1, \mathcal{R}_1 \rangle$ *and the indexed predicate* $\Pi 1$ *satisfy the conditions (LP), (∃P) and (∀P) of Section 7.2.*

Proof. Since the aim of this first example is to illustrate the proof technique, we provide a proof where we explain the reasoning in detail.

Proof of (LP): We should prove this condition for each (instantiation of meta-)rule in Fig. 1.

Case: (APP) Assume that $\vdash e_1\ e_2 : T$ holds. We have to find types for the premises. We proceed as follows:
(1) First premise: by Lemma 8.1 (4), $\vdash e_1 : T' \to T$.
(2) Second premise: again by Lemma 8.1 (4), $\vdash e_2 : T'$ (without needing the assumption $\vdash \lambda x.e : T' \to T$).
(3) Third premise: $\vdash e[v_2/x] : T$ should hold (assuming $\vdash \lambda x.e : T' \to T$, $\vdash v_2 : T'$). Since $\vdash \lambda x.e : T' \to T$, by Lemma 8.1 (3) we have $x{:}T' \vdash e : T$, so by Lemma 8.2 and $\vdash v_2 : T'$ we have $\vdash e[v_2/x] : T$.

Finally, we have to show $\vdash v : T$, assuming $\vdash \lambda x.e : T' \to T$, $\vdash v_2 : T'$ and $\vdash v : T$, which is trivial from the third assumption.

Case: (SUCC) Assume that $\vdash \mathtt{succ}\ e : T$ holds. By Lemma 8.1 (5), $T = \mathtt{Nat}$, and $\vdash e : \mathtt{Nat}$, hence we find $\mathtt{Nat}$ as type for the premise. Moreover, $\vdash n + 1 : \mathtt{Nat}$ holds by rule (T-CONST).

Case: (CHOICE) Assume that $\vdash e_1 \oplus e_2 : T$ holds. By Lemma 8.1 (6), we have $\vdash e_i : T$, with $i \in 1, 2$. Hence we find T as type for the premise. Finally, we have to show $\vdash v : T$, assuming $\vdash v : T$, which is trivial.

Case: (VAL) Trivial by assumption.

Proof of (∃P): We should prove that, for each configuration (here, expression e) such that $\vdash e : T$ holds for some T, there is a rule with this configuration in the conclusion. The expression e cannot be a variable, since a variable cannot be typed in the empty environment. Application, successor, choice, abstraction and constants appear as consequence in the big-step rules (APP), (SUCC), (CHOICE) and (VAL).

Proof of (∀P): We should prove this condition for each (instantiation of meta-)rule.

Case: (APP) Assuming $\vdash e_1\ e_2 : T$, again by Lemma 8.1 (4) we get $\vdash e_1 : T' \to T$.

(1) First premise: if $e_1 \Rightarrow v$ is derivable, then there should be a rule with $e_1\ e_2$ in the conclusion and $e_1 \Rightarrow v$ as first premise. Since we proved (LP), by preservation (Lemma 7.2) $\vdash v : T' \to T$ holds. Then, by Lemma 8.3 (1), v has shape $\lambda x.e$, hence the required rule exists. As noted at page 27, in practice checking (∀P) for a (meta-)rule amounts to show that configurations in the premises evaluate to results which have the required shape (to be a λ-abstraction in this case).

(2) Second premise: if $e_1 \Rightarrow \lambda x.e$, and $e_2 \Rightarrow v$, then there should be a rule with $e_1\ e_2$ in the conclusion and $e_1 \Rightarrow \lambda x.e$, $e_2 \Rightarrow v$ as first two premises. This is trivial since the meta-variable v_2 can be freely instantiated in the meta-rule.

(3) Third premise: trivial as the previous one.

Case: (SUCC) Assuming $\vdash \mathtt{succ}\ e : T$, again by Lemma 8.1 (5) we get $\vdash e : \mathtt{Nat}$. If $e \Rightarrow v$ is derivable, there should be a rule with $\mathtt{succ}\ e$ in the conclusion and $e \Rightarrow v$ as first premise. Indeed, by preservation (Lemma 7.2) and Lemma 8.3 (2), v has shape n.

Case: (CHOICE) Trivial since the meta-variable v can be freely instantiated.

Case: (VAL) Empty, because there are no premises.

□

An interesting remark is that, differently from the standard approach, there is *no induction* in the proof: everything is *by cases*. This is a consequence of the fact that, as discussed in Section 7.2, the three conditions are *local*, that is, they are conditions on single rules. Induction is "hidden" once and for all in the proof that those three conditions are sufficient to ensure soundness.

If we drop in Fig. 1 rule (SUCC), then condition (∃P) fails, since there is no longer a rule for the well-typed configuration $\mathtt{succ}\ n$. If we add the (FOOL) rule $\vdash 0\ 0 : \mathtt{Nat}$, then condition (∀P) fails for rule (APP), since $0 \Rightarrow 0$ is derivable, but there is no rule with $0\ 0$ in the conclusion and $0 \Rightarrow 0$ as first premise.

8.2 MiniFJ&λ

In this example, the language is a subset of FJ&λ [13], a calculus extending Featherweight Java (FJ) with λ-abstractions and intersection types, introduced in Java 8. To keep the example small, we do not consider intersections and focus on one key typing feature: λ-abstractions can only be typed when occurring in a context requiring a given type (called the *target type*). In a small-step semantics, this poses a problem: reduction can move λ-abstractions into arbitrary contexts, leading to intermediate terms which would be ill-typed. To maintain subject reduction, Bettini et al. [13] decorate λ-abstractions with their initial target type. In a big-step semantics, there is no need of intermediate terms and annotations.

The syntax is given in the first part of Fig. 9. We assume sets of *variables* x, *class names* C, *interface names* I, J, *field names* f, and *method names* m. As usual, we assume a special variable `this`, used in method bodies to refer to the receiver object. Interfaces which have *exactly* one method (dubbed *functional interfaces*) can be used as target types.

$$
\begin{array}{rcll}
e & ::= & x \mid e.\mathsf{f} \mid \mathsf{new}\ \mathsf{C}(e_1, \ldots, e_n) \mid e.\mathsf{m}(e_1, \ldots, e_n) \mid \lambda xs.e \mid (T)e & \text{expression} \\
T & ::= & \mathsf{C} \mid \mathsf{I} & \text{type}
\end{array}
$$

$$
\begin{array}{rcll}
c & ::= & \langle \mathrm{E}, e \rangle & \text{configuration} \\
v & ::= & [vs]^{\mathsf{C}} \mid \lambda xs.e & \text{result (value)}
\end{array}
$$

$$
(\textsc{var})\ \frac{}{\langle \mathrm{E}, x \rangle \Rightarrow v}\ \mathrm{E}(x) = v \qquad (\textsc{new})\ \frac{\langle \mathrm{E}, e_i \rangle \Rightarrow v_i \quad \forall i \in 1..n}{\langle \mathrm{E}, \mathsf{new}\ \mathsf{C}(e_1, \ldots, e_n) \rangle \Rightarrow [v_1, \ldots, v_n]^{\mathsf{C}}}
$$

$$
(\textsc{field-access})\ \frac{\langle \mathrm{E}, e \rangle \Rightarrow [v_1, \ldots, v_n]^{\mathsf{C}}}{\langle \mathrm{E}, e.\mathsf{f}_i \rangle \Rightarrow v_i}\ \begin{array}{l} \mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1\,; \ldots T_n\,\mathsf{f}_n\,; \\ i \in 1..n \end{array}
$$

$$
(\textsc{invk})\ \frac{\begin{array}{l} \langle \mathrm{E}, e_0 \rangle \Rightarrow [vs]^{\mathsf{C}} \\ \langle \mathrm{E}, e_i \rangle \Rightarrow v_i \quad \forall i \in 1..n \\ \langle x_1{:}v_1, \ldots, x_n{:}v_n, \mathsf{this}{:}[vs]^{\mathsf{C}}, e \rangle \Rightarrow v \end{array}}{\langle \mathrm{E}, e_0.\mathsf{m}(e_1, \ldots, e_n) \rangle \Rightarrow v}\ \mathsf{mbody}(\mathsf{C}, \mathsf{m}) = \langle x_1 \ldots x_n, e \rangle
$$

$$
(\lambda\textsc{-invk})\ \frac{\begin{array}{l} \langle \mathrm{E}, e_0 \rangle \Rightarrow \lambda x_1 \ldots x_n.e \\ \langle \mathrm{E}, e_i \rangle \Rightarrow v_i \quad \forall i \in 1..n \\ \langle x_1{:}v_1, \ldots, x_n{:}v_n, e \rangle \Rightarrow v \end{array}}{\langle \mathrm{E}, e_0.\mathsf{m}(e_1, \ldots, e_n) \rangle \Rightarrow v}
$$

$$
(\lambda)\ \frac{}{\langle \mathrm{E}, \lambda xs.e \rangle \Rightarrow \lambda xs.e} \qquad (\textsc{upcast})\ \frac{\langle \mathrm{E}, e \rangle \Rightarrow v}{\langle \mathrm{E}, (T)e \rangle \Rightarrow v}
$$

Fig. 9. MiniFJ&λ: syntax and big-step semantics

Expressions are those of FJ, plus λ-abstractions, and types are class and interface names. Throughout this section xs and vs denote lists of variables and values, respectively. In $\lambda xs.e$ we assume that xs is not empty and e is not a λ-abstraction. For simplicity, we only consider *upcasts*, which have no runtime effect, but are important to allow the programmer to use λ-abstractions, as exemplified in discussing typing rules.

To be concise, the class table is abstractly modelled as follows:

- $\mathsf{fields}(\mathsf{C})$ gives the sequence of field declarations $T_1\,\mathsf{f}_1\,; .. T_n\,\mathsf{f}_n\,;$ for class C
- $\mathsf{mtype}(T, \mathsf{m})$ gives, for each method m in class or interface T, the pair $T_1 \ldots T_n \to T'$ consisting of the parameter types and return type
- $\mathsf{mbody}(\mathsf{C}, \mathsf{m})$ gives, for each method m in class C, the pair $\langle x_1 \ldots x_n, e \rangle$ consisting of the parameters and body
- $<:$ is the reflexive and transitive closure of the union of the extends and implements relations, stating that two class or interface names are related iff they occur in the class table connected by the keywords extends or implements
- $!\mathsf{mtype}(\mathsf{I})$ gives, for each *functional* interface I, $\mathsf{mtype}(\mathsf{I}, \mathsf{m})$, where m is the only method of I.

The big-step semantics is given in the last part of Fig. 9. MiniFJ&λ shows an example of instantiation of the framework where configurations include an auxiliary structure, rather than being just language terms. In this case, the structure is an *environment* E (a finite map from variables to values) modelling the current stack frame. Furthermore, results are not particular configurations: they are either *objects*, of shape $[vs]^{\mathsf{C}}$, or λ-abstractions.

Rules for FJ constructs are straightforward. Note that, since we only consider upcasts, casts have no runtime effect. Indeed, they are guaranteed to succeed on well-typed expressions. Rule (λ-invk) shows that, when the receiver of a

$$\text{(T-CONF)}\ \frac{\vdash v_i : T_i \quad \forall i \in 1..n \qquad x_1{:}T'_1, \ldots, x_n{:}T'_n \vdash e : T}{\vdash \langle x_1{:}v_1, \ldots, x_n{:}v_n, e\rangle : T} \quad T_i <: T'_i \quad \forall i \in 1..n$$

$$\text{(T-VAR)}\ \frac{}{\Gamma \vdash x : T} \quad \Gamma(x) = T \qquad \text{(T-UPCAST)}\ \frac{\Gamma \vdash e : T}{\Gamma \vdash (T)e : T}$$

$$\text{(T-FIELD-ACCESS)}\ \frac{\Gamma \vdash e : \mathsf{C}}{\Gamma \vdash e.\mathsf{f} : T_i} \quad \begin{array}{l}\mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1\,; \ldots T_n\,\mathsf{f}_n\,; \\ i \in 1..n\end{array}$$

$$\text{(T-NEW)}\ \frac{\Gamma \vdash e_i : T_i \quad \forall i \in 1..n}{\Gamma \vdash \mathsf{new}\,\mathsf{C}(e_1, \ldots, e_n) : \mathsf{C}} \quad \mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1\,; \ldots T_n\,\mathsf{f}_n\,;$$

$$\text{(T-INVK)}\ \frac{\Gamma \vdash e_i : T_i \quad \forall i \in 0..n}{\Gamma \vdash e_0.\mathsf{m}(e_1, \ldots, e_n) : T} \quad \begin{array}{l}e_0 \text{ not of shape } \lambda xs.e \\ \mathsf{mtype}(T_0, \mathsf{m}) = T_1 \ldots T_n \to T\end{array}$$

$$\text{(T-}\lambda\text{)}\ \frac{x_1{:}T_1, \ldots, x_n{:}T_n \vdash e : T}{\Gamma \vdash \lambda x_1 \ldots x_n.e : \mathsf{I}} \quad !\mathsf{mtype}(\mathsf{I}) = T_1 \ldots T_n \to T$$

$$\text{(T-OBJECT)}\ \frac{\Gamma \vdash v_i : T'_i \quad \forall i \in 1..n}{\Gamma \vdash [v_1, \ldots, v_n]^{\mathsf{C}} : \mathsf{C}} \quad \begin{array}{l}\mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1\,; \ldots T_n\,\mathsf{f}_n\,; \\ T'_i <: T_i \quad \forall i \in 1..n\end{array}$$

$$\text{(T-SUB)}\ \frac{\Gamma \vdash e : T}{\Gamma \vdash e : T'} \quad \begin{array}{l}e \text{ not of shape } \lambda xs.e \\ T <: T'\end{array}$$

Fig. 10. MiniFJ&λ: type system

method is a λ-abstraction, the method name is not significant at runtime, and the effect is that the body of the function is evaluated as in the usual application.

The type system, consisting of judgements for configurations, expressions and values, is given in Fig. 10. The following assumptions formalize standard FJ typing constraints on the class table.

(FJ1) Method bodies are well-typed with respect to method types:
- either $\mathsf{mbody}(\mathsf{C}, \mathsf{m})$ and $\mathsf{mtype}(\mathsf{C}, \mathsf{m})$ are both undefined
- or $\mathsf{mbody}(\mathsf{C}, \mathsf{m}) = \langle x_1 \ldots x_n, e\rangle$, $\mathsf{mtype}(\mathsf{C}, \mathsf{m}) = T_1 \ldots T_n \to T$, and $x_1{:}T_1, \ldots, x_n{:}T_n, \mathtt{this}{:}\mathsf{C} \vdash e : T$.

(FJ2) Fields are inherited, no field hiding:
if $T <: T'$, and $\mathsf{fields}(T') = T_1\,\mathsf{f}_1\,; \ldots T_n\,\mathsf{f}_n\,;$, then $\mathsf{fields}(T) = T_1\,\mathsf{f}_1\,; \ldots T_m\,\mathsf{f}_m\,;$, $m \geq n$, and $\mathsf{f}_i \neq \mathsf{f}_j$ for $i \neq j$.

(FJ3) Methods are inherited, no method overloading, invariant overriding:
if $T <: T'$, and $\mathsf{mtype}(T', \mathsf{m})$ is defined, then $\mathsf{mtype}(T, \mathsf{m}) = \mathsf{mtype}(T', \mathsf{m})$.

Besides the standard typing features of FJ, the MiniFJ&λ type system ensures the following.

- A functional interface I can be assigned as type to a λ-abstraction which has the functional type of the method, see rule (T-λ).
- A λ-abstraction should have a *target type* determined by the context where the λ-abstraction occurs. More precisely, as described by Gosling et al. [31, p. 602], a λ-abstraction in our calculus can only occur as return expression of a method or argument of constructor, method call or cast. Then, in some contexts a λ-abstraction cannot be typed, in our calculus when occurring as receiver in field access or method invocation, hence these cases should be prevented. This is implicit in rule (T-FIELD-ACCESS), since the type of the receiver should be a class name, whereas it is explicitly forbidden in rule (T-INVK). Finally, a λ-abstraction cannot be the main expression of a

program, as also in this case the target type is not well defined. For simplicity, this requirement is not enforced by typing rules, but it can be easily recovered as an assumption on the source program.

- A λ-abstraction with a given target type J should have type *exactly* J: a subtype I of J is not enough. Consider, for instance, the following class table:

```
interface J {}
interface I extends J { A m(A x); }
class C { J f; }
class D {
  D m(I y) { return new D().n(y); }
  D n(J y) { return new D(); }
}
```

  In the main expression $\mathsf{new}\ \mathsf{D}().\mathsf{n}(\lambda x.x)$, the λ-abstraction has target type J, which is *not* a functional interface, hence the expression is ill-typed in Java (the compiler has no functional type against which to typecheck the λ-abstraction). On the other hand, in the body of method m, the parameter y of type I can be passed, as usual, to method n expecting a supertype. For instance, the main expression $\mathsf{new}\ \mathsf{D}().\mathsf{m}(\lambda x.x)$ is well-typed, since the λ-abstraction has target type I, and can be safely passed to method n, since it is not used as function there. To formalise this behaviour, it is forbidden to apply subsumption to λ-abstractions, see rule (T-SUB).
- However, λ-abstractions occurring as results rather than in source code (that is, in the environment and as fields of objects) are allowed to have a subtype of the required type, see the explicit side condition in rules (T-CONF) and (T-OBJECT). For instance, in the above class table, the expression $\mathsf{new}\ \mathsf{C}((\mathsf{I})\lambda x.x)$ is well-typed, whereas $\mathsf{new}\ \mathsf{C}(\lambda x.x)$ is ill typed, since rule (T-SUB) cannot be applied to λ-abstractions. When the expression is evaluated, the result is $[\lambda x.x]^{\mathsf{C}}$, which is well-typed.

As mentioned at the beginning, the obvious small-step semantics would produce not typable expressions. In the above example, we get

$$\mathsf{new}\ \mathsf{C}((\mathsf{I})\lambda x.x) \longrightarrow \mathsf{new}\ \mathsf{C}(\lambda x.x) \longrightarrow [\lambda x.x]^{\mathsf{C}}$$

and $\mathsf{new}\ \mathsf{C}(\lambda x.x)$ has no type, while $\mathsf{new}\ \mathsf{C}((\mathsf{I})\lambda x.x)$ and $[\lambda x.x]^{\mathsf{C}}$ have type C.

As expected, to show soundness (Theorem 8.7) lemmas of inversion and canonical forms are handy: they can be easily proved as usual. Instead, we do not need a substitution lemma, since environments associate variables with values.

LEMMA 8.5 (INVERSION). *The following hold:*

(1) If $\vdash \langle x_1{:}v_1, \ldots, x_n{:}v_n, e\rangle : T$, *then* $x_1{:}T_1, \ldots, x_n{:}T_n \vdash e : T$, $\vdash v_i : T_i'$ *and* $T_i' <: T_i$ *for all* $i \in 1..n$.
(2) If $\Gamma \vdash x : T$, *then* $\Gamma(x) <: T$.
(3) If $\Gamma \vdash e.\mathsf{f}_i : T$, *then* $\Gamma \vdash e : \mathsf{C}$ *and* $\mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1; \ldots T_n\,\mathsf{f}_n;$ *and* $T_i <: T$ *where* $i \in 1..n$.
(4) If $\Gamma \vdash \mathsf{new}\ \mathsf{C}(e_1, \ldots, e_n) : T$, *then* $\mathsf{C} <: T$ *and* $\mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1; \ldots T_n\,\mathsf{f}_n;$ *and* $\Gamma \vdash e_i : T_i$ *for all* $i \in 1..n$.
(5) If $\Gamma \vdash e_0.\mathsf{m}(e_1, \ldots, e_n) : T$, *then* e_0 *not of shape* $\lambda xs.e$ *and* $\Gamma \vdash e_i : T_i$ *for all* $i \in 0..n$ *and* $\mathsf{mtype}(T_0, \mathsf{m}) = T_1 \ldots T_n \to T'$ *with* $T' <: T$.
(6) If $\Gamma \vdash \lambda xs.e : T$, *then* $T = \mathsf{I}$ *and* $!\mathsf{mtype}(\mathsf{I}) = T_1 \ldots T_n \to T'$ *and* $x_1{:}T_1, \ldots, x_n{:}T_n \vdash e : T'$.
(7) If $\Gamma \vdash (T')e : T$, *then* $\Gamma \vdash e : T'$ *and* $T' <: T$.
(8) If $\Gamma \vdash [v_1, \ldots, v_n]^{\mathsf{C}} : T$, *then* $\mathsf{C} <: T$ *and* $\mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1; \ldots T_n\,\mathsf{f}_n;$ *and* $\Gamma \vdash v_i : T_i'$ *and* $T_i' <: T_i$ *for all* $i \in 1..n$.

Lemma 8.6 (Canonical Forms). *The following hold:*

(1) If $\vdash v : \mathsf{C}$, *then* $v = [vs]^{\mathsf{D}}$ *and* $\mathsf{D} <: \mathsf{C}$.

(2) If $\vdash v : \mathsf{I}$, *then either* $v = [vs]^{\mathsf{C}}$ *and* $\mathsf{C} <: \mathsf{I}$ *or* $v = \lambda xs.e$ *and* I *is a functional interface.*

We write $\Gamma \vdash e :<: T$ as short for $\Gamma \vdash e : T'$ and $T' <: T$ for some T'. In order to state soundness, set $\langle C_2, R_2, \mathcal{R}_2 \rangle$ the big-step semantics defined in Fig. 9, and let $\Pi 2^C_T = \{\langle \mathrm{E}, e\rangle \in C_2 \mid \vdash \langle \mathrm{E}, e\rangle :<: T\}$ and $\Pi 2^R_T = \{v \in R_2 \mid \vdash v :<: T\}$, for T defined in Fig. 9.

Theorem 8.7 (Soundness). *The big-step semantics* $\langle C_2, R_2, \mathcal{R}_2 \rangle$ *and the indexed predicate* $\Pi 2$ *satisfy the conditions (LP), (∃P) and (∀P) of Section 7.2.*

Proof. *Proof of (LP)*: The proof is by cases on instantiations of meta-rules. In all such cases, we have a configuration $\langle \mathrm{E}, e\rangle$ in the conclusion, with $\mathrm{E} = y_1{:}\hat{v}_1, \ldots, y_p{:}\hat{v}_p$, such that $\vdash \langle y_1{:}\hat{v}_1, \ldots, y_p{:}\hat{v}_p, e\rangle :<: \hat{T}$, hence, by Lemma 8.5 (1), we get $\vdash \hat{v}_\ell :<: \hat{T}_\ell$ for all $\ell \in 1..p$ and $\Gamma \vdash e : T$ with $\Gamma = y_1{:}\hat{T}_1, \ldots, y_p{:}\hat{T}_p$ and $T <: \hat{T}$, for some $\hat{T}_1, \ldots, \hat{T}_p$.

Case: (VAR) Lemma 8.5 (2) applied to $\Gamma \vdash x : T$ implies $x = y_i$ and $\hat{T}_i <: T$ for some $i \in 1..p$. Then, the thesis follows by transitivity of subtyping since $\mathrm{E}(x) = \hat{v}_i$ and $\vdash \hat{v}_i :<: \hat{T}_i$.

Case: (FIELD-ACCESS) Lemma 8.5 (3) applied to $\Gamma \vdash e.\mathsf{f}_i : T$ implies $\Gamma \vdash e : \mathsf{D}$ and $\mathsf{fields}(\mathsf{D}) = T_1\,\mathsf{f}_1; \ldots T_m\,\mathsf{f}_m;$ and $T_i <: T$ where $i \in 1..m$. Since $\langle \mathrm{E}, e\rangle \Rightarrow [v_1, \ldots, v_n]^{\mathsf{C}}$ is a premise we assume $\vdash [v_1, \ldots, v_n]^{\mathsf{C}} :<: \mathsf{D}$, which implies $\mathsf{C} <: \mathsf{D}$ and $\mathsf{fields}(\mathsf{C}) = T'_1\,\mathsf{f}'_1; \ldots T'_n\,\mathsf{f}'_n;$ and $\Gamma \vdash v_j :<: T'_j$ for all $j \in 1..n$ by Lemma 8.5 (8). From $\mathsf{C} <: \mathsf{D}$ and assumption (FJ2) we have $m \le n$ and $T_j = T'_j$ and $\mathsf{f}_j = \mathsf{f}'_j$ for all $j \in 1..m$. We conclude $\vdash v_i :<: T$.

Case: (NEW) Lemma 8.5 (4) applied to $\Gamma \vdash \mathtt{new}\ \mathsf{C}(e_1, \ldots, e_n) : T$ implies $\mathsf{C} <: T$ and $\mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1; \ldots T_n\,\mathsf{f}_n;$ and $\Gamma \vdash e_i : T_i$ for all $i \in 1..n$. Since $\langle \mathrm{E}, e_i\rangle \Rightarrow v_i$ is a premise we assume $\vdash v_i :<: T_i$ for all $i \in 1..n$. Using rule (T-OBJECT) we derive $\vdash [v_1, \ldots, v_n]^{\mathsf{C}} :<: T$.

Case: (INVK) Lemma 8.5 (5) applied to $\Gamma \vdash e_0.\mathsf{m}(e_1, \ldots, e_n) : T$ implies e_0 not of shape $\lambda xs.e$ and $\Gamma \vdash e_i : T_i$ for all $i \in 0..n$ and $\mathsf{mtype}(T_0, \mathsf{m}) = T_1 \ldots T_n \to T'$ with $T' <: T$. Since $\langle \mathrm{E}, e_0\rangle \Rightarrow [vs']^{\mathsf{C}}$ is a premise we assume $\vdash [vs']^{\mathsf{C}} :<: T_0$, which implies $\mathsf{C} <: T_0$ by Lemma 8.5 (8). Since $\langle \mathrm{E}, e_i\rangle \Rightarrow v_i$ is a premise we assume $\vdash v_i :<: T_i$ for all $i \in 1..n$. We have $\mathsf{mtype}(\mathsf{C}, \mathsf{m}) = T_1 \ldots T_n \to T'$ since $\mathsf{mtype}(T_0, \mathsf{m}) = T_1 \ldots T_n \to T'$ and $\mathsf{C} <: T_0$ by assumption (FJ3). By assumption (FJ1), $x_1{:}T_1, \ldots, x_n{:}T_n, \mathtt{this}{:}\mathsf{C} \vdash e : T'$. Therefore, by rule (T-CONF) and since $T' <: T$, we can derive $\vdash \langle x_1{:}v_1, \ldots, x_n{:}v_n, \mathtt{this}{:}[vs']^{\mathsf{C}}, e\rangle :<: T$.

Case: (λ-INVK) Lemma 8.5 (5) applied to $\Gamma \vdash e_0.\mathsf{m}(e_1, \ldots, e_n) : T$ implies $\Gamma \vdash e_i : T_i$ for all $i \in 0..n$ and $\mathsf{mtype}(T_0, \mathsf{m}) = T_1 \ldots T_n \to T'$ with $T' <: T$. Since $\langle \mathrm{E}, e_0\rangle \Rightarrow \lambda xs.e$ is a premise we assume $\vdash \lambda xs.e :<: T_0$, which implies $\mathsf{I} <: T_0$ and $!\mathsf{mtype}(\mathsf{I}) = T_1 \ldots T_n \to T'$ and $x_1{:}T_1, \ldots, x_n{:}T_n \vdash e : T'$ by Lemma 8.5 (6). Since $\langle \mathrm{E}, e_i\rangle \Rightarrow v_i$ is a premise we assume $\vdash v_i :<: T_i$ for all $i \in 1..n$. Therefore we derive $\vdash \langle x_1{:}v_1, \ldots, x_n{:}v_n, e\rangle :<: T$.

Case: (λ) The thesis is trivial as the configuration and the final result are the same.

Case: (UPCAST) Lemma 8.5 (7) applied to $\Gamma \vdash (T')e : T$ implies $\Gamma \vdash e :<: T$. From $\langle \mathrm{E}, e\rangle \Rightarrow v$ we conclude $\vdash v :<: T$.

Proof of (∃P): It is easy to verify that if $\vdash \langle \mathrm{E}, e\rangle :<: T$, then there is a rule in Fig. 9, whose conclusion is $\langle \mathrm{E}, e\rangle$, just because for every syntactic construct there is a corresponding rule and side conditions in typing rules imply those of big-step rules. The only less trivial case is that of variables: if $\vdash \langle \mathrm{E}, x\rangle :<: T$, then by Lemma 8.5 (1,2), $x \in \mathsf{dom}(\mathrm{E})$, hence rule (VAR) is applicable, as the side condition is satisfied.

Proof of (∀P): Rule (FIELD-ACCESS) requires that $\langle \mathrm{E}, e\rangle$ reduces to an object with a field f_i, and this is assured by the typing rule (T-FIELD-ACCESS), which prescribes a class type for the expression e with the field f_i, together with the validity of condition

$$T \quad ::= \quad \mathtt{Nat} \mid T_1 \to T_2 \mid T_1 \wedge T_2 \mid T_1 \vee T_2 \qquad \text{type}$$

$$(\wedge\,\mathrm{I})\ \frac{\Gamma \vdash e : T \quad \Gamma \vdash e : S}{\Gamma \vdash e : T \wedge S} \qquad (\wedge\,\mathrm{E})\ \frac{\Gamma \vdash e : T \wedge S}{\Gamma \vdash e : T} \qquad (\wedge\,\mathrm{E})\ \frac{\Gamma \vdash e : T \wedge S}{\Gamma \vdash e : S}$$

$$(\vee\,\mathrm{I})\ \frac{\Gamma \vdash e : T}{\Gamma \vdash e : T \vee S} \qquad (\vee\,\mathrm{I})\ \frac{\Gamma \vdash e : S}{\Gamma \vdash e : T \vee S}$$

Fig. 11. Intersection and union types: syntax and typing rules

(LP) (which assures type preservation by Lemma 7.2) and Lemma 8.6 (1). For a well-typed method call $e_0 . \mathtt{m}(e_1, \ldots, e_n)$ the configuration $\langle \mathrm{E}, e_0 \rangle$ can reduce either to an object or to a λ-expression. In the first case we can apply rule (INVK) and in the second case rule (λ-INVK). In both cases the typing assures that the arguments are in the right number, while the condition is trivial for the last premise. □

8.3 Intersection and union types

We enrich the type system of Fig. 8 by adding intersection and union type constructors and the corresponding typing rules, see Fig. 11. Intersection types for the λ-calculus have been widely studied, *e.g.*, by Barendregt et al. [12]. Union types naturally model conditionals [32] and non-deterministic choice [30].

The production in the top section of Fig. 11 is again interpreted coinductively to allow possibly infinite types, but, as usual with recursive types, we only consider *contractive* types [44], that is, we require an infinite number of arrows in each infinite path in a type (viewed as a tree). On the other hand, typing rules are still interpreted inductively.

The typing rules for the introduction and the elimination of intersection and union are standard, except for the absence of the union elimination rule:

$$(\vee E)\ \frac{\Gamma\{T/x\} \vdash e : V \quad \Gamma\{S/x\} \vdash e : V \quad \Gamma \vdash e' : T \vee S}{\Gamma \vdash e[e'/x] : V}$$

As a matter of fact, rule ($\vee E$) is unsound for $\oplus$. For example, let split the type Nat into Even and Odd and add the expected typings for natural numbers. The prefix addition + has type $(\mathtt{Even} \to \mathtt{Even} \to \mathtt{Even}) \wedge (\mathtt{Odd} \to \mathtt{Odd} \to \mathtt{Even})$ and we derive

$$\dfrac{x{:}\mathtt{Even} \vdash +\,x\,x{:}\mathtt{Even} \quad x{:}\mathtt{Odd} \vdash +\,x\,x{:}\mathtt{Even} \qquad \dfrac{\dfrac{}{\vdash 1 : \mathtt{Odd}}}{\vdash 1 : \mathtt{Even} \vee \mathtt{Odd}} \quad \dfrac{\dfrac{}{\vdash 2 : \mathtt{Even}}}{\vdash 2 : \mathtt{Even} \vee \mathtt{Odd}}}{\vdash (1 \oplus 2) : \mathtt{Even} \vee \mathtt{Odd}}}{\vdash +(1 \oplus 2)\,(1 \oplus 2) : \mathtt{Even}}$$

We cannot assign the type Even to 3, which is a possible result, so strong soundness is lost. In addition, in the small-step approach, we cannot assign Even to the intermediate term + 1 2, so subject reduction fails. In the big-step approach, there is no such intermediate term; however, condition (LP) fails for the big-step rule for +. Indeed, considering the following instantiation of the rule:

$$(+)\ \frac{1 \oplus 2 \Rightarrow 1 \quad 1 \oplus 2 \Rightarrow 2}{+(1 \oplus 2)(1 \oplus 2) \Rightarrow 3}$$

and the type Even for the conclusion: we cannot assign this type to the final result as required by (LP) (cf. Definition 7.1 (2)).

Intersection types allow to derive meaningful types also for expressions containing variables applied to themselves, for example we can derive $\vdash \lambda x.x\,x : (T \to S) \wedge T \to S$. With union types all non-deterministic choices between typable expressions can be typed too, since we can derive $\Gamma \vdash e_1 \oplus e_2 : T_1 \vee T_2$ from $\Gamma \vdash e_1 : T_1$ and $\Gamma \vdash e_2 : T_2$.

We now state standard lemmas for the type system, which are handy towards the soundness proof. We first define the *subtyping* relation $T \leq S$ as the smallest preorder such that:

- $S \le T_1$ and $S \le T_2$ imply $S \le T_1 \wedge T_2$;
- $T \wedge S \le T$ and $T \wedge S \le S$;
- $T \le T \vee S$ and $T \le S \vee T$.

It is easy to verify that $T \le S$ iff $\Gamma, x{:}T \vdash x : S$ for an arbitrary variable x, using rules (∧I), (∧E) and (∨I).

Lemma 8.8 (Inversion). *The following hold:*

(1) If $\Gamma \vdash x : T$, *then* $\Gamma(x) \le T$.
(2) If $\Gamma \vdash n : T$, *then* $\mathtt{Nat} \le T$.
(3) If $\Gamma \vdash \lambda x.e : T$, *then* $\Gamma\{S_i/x\} \vdash e : V_i$ *for* $i \in 1..m$ *and* $\bigwedge_{i\in 1..m}(S_i \to V_i) \le T$.
(4) If $\Gamma \vdash e_1\ e_2 : T$, *then* $\Gamma \vdash e_1 : S_i \to V_i$ *and* $\Gamma \vdash e_2 : S_i$ *for* $i \in 1..m$ *and* $\bigwedge_{i\in 1..m} V_i \le T$.
(5) If $\Gamma \vdash \mathtt{succ}\ e : T$, *then* $\mathtt{Nat} \le T$ *and* $\Gamma \vdash e : \mathtt{Nat}$.
(6) If $\Gamma \vdash e_1 \oplus e_2 : T$, *then* $\Gamma \vdash e_i : T'$ *with* $T' \le T$ *and* $i \in 1..2$.

Lemma 8.9 (Substitution). *If* $\Gamma\{T'/x\} \vdash e : T$ *and* $\Gamma \vdash e' : T'$, *then* $\Gamma \vdash e[e'/x] : T$.

Lemma 8.10 (Canonical Forms). *The following hold:*

(1) If $\vdash v : T' \to T$, *then* $v = \lambda x.e$.
(2) If $\vdash v : \mathtt{Nat}$, *then* $v = n$.

In order to state soundness, let $\Pi 3^C_T = \{e \in C_1 \mid\vdash e : T\}$ and $\Pi 3^R_T = \{v \in R_1 \mid\vdash v : T\}$, for T defined in Fig. 11.

Theorem 8.11 (Soundness). *The big-step semantics* $\langle C_1, R_1, \mathcal{R}_1\rangle$ *and the indexed predicate* $\Pi 3$ *satisfy the conditions (lp), (∃p) and (∀p) of Section 7.2.*

Proof Sketch. We prove conditions only for rule (app), the other cases are similar (cf. proof of Theorem 8.4).

Proof of (lp): The proof is by cases on instantiations of meta-rules. For rule (app) Lemma 8.8 (4) applied to $\vdash e_1\ e_2 : T$ implies $\vdash e_1 : S_i \to V_i$ and $\vdash e_2 : S_i$ for $i \in 1..m$ and $\bigwedge_{i\in 1..m} V_i \le T$. Now, from assumptions of (lp), we get $\vdash \lambda x.e : S_i \to V_i$ and $\vdash v_2 : S_i$ for $i \in 1..m$. Lemma 8.8 (3) implies $x : S_i \vdash e : V_i$, so by Lemma 8.9 we have $\vdash e[v_2/x] : V_i$ for $i \in 1..m$. We can derive $\vdash e[v_2/x] : T$ using rules (∧I), (∧E) and (∨I).

Proof of (∃p): The proof is as in Theorem 8.4.

Proof of (∀p): The proof is by cases on instantiations of meta-rules. For rule (app) Lemma 8.8 (4) applied to $\vdash e_1\ e_2 : T$ implies $\vdash e_1 : S_i \to V_i$ for $i \in 1..m$. If $e_1 \Rightarrow v$ we get $\vdash v : S_i \to V_i$ for $i \in 1..m$ by (lp) and Lemma 7.2. Lemma 8.10 (1) applied to $\vdash v : S_i \to V_i$ implies $v = \lambda x.e$ as needed. □

8.4 MiniFJ$^\vee$

A well-known example in which proving soundness with respect to small-step semantics is extremely challenging is the standard type system with intersection and union types [11] w.r.t. the pure λ-calculus with full reduction. Indeed, the standard subject reduction technique fails[8], since, for instance, we can derive the type

$$(T \to T \to V) \wedge (S \to S \to V) \to (U \to T \vee S) \to U \to V$$

[8]For this reason, Barbanera et al. [11] prove soundness by an ad-hoc technique, that is, by considering parallel reduction and an equivalent type system à la Gentzen, which enjoys the cut elimination property.

for both $\lambda x.\lambda y.\lambda z.x\ ((\lambda t.t)\ (y\ z))\ ((\lambda t.t)\ (y\ z))$ and $\lambda x.\lambda y.\lambda z.x\ (y\ z)\ (y\ z)$, but the intermediate expressions $\lambda x.\lambda y.\lambda z.x\ ((\lambda t.t)\ (y\ z))\ (y\ z)$ and $\lambda x.\lambda y.\lambda z.x\ (y\ z)\ ((\lambda t.t)\ (y\ z))$ do not have this type.

As the example shows, the key problem is that rule (∨E) can be applied to expression e where the same subexpression e' occurs more than once. In the non-deterministic case, as shown by the example in the previous section, this is unsound, since e' can reduce to different values. In the deterministic case, instead, this is sound, but cannot be proved by subject reduction. Since using big-step semantics there are no intermediate steps to be typed, our approach seems very promising to investigate an alternative proof of soundness. Whereas we leave this challenging problem to future work, here as first step we describe a calculus with a much simpler version of the problematic feature.

The calculus is a variant of FJ∨, introduced by Igarashi and Nagira [33], an extension of FJ [34] with union types. As discussed more extensively by Igarashi and Nagira [33], this gives the ability to define a supertype even after a class hierarchy is fixed, grouping independently developed classes with similar interfaces. In fact, given some types, their union type can be viewed as an interface type that "factors out" their common features. With respect to FJ∨, we do not consider cast and type-case constructs and, more importantly, in the typing rules we handle differently union types, taking inspiration directly from rule (∨E) of the λ-calculus. With this approach, we enhance the expressivity of the type system, since it becomes possible to eliminate unions simultaneously for an arbitrary number of arguments, including the receiver, in a method invocation, provided that they are all equal to each other. We dub this calculus MiniFJ$^{\vee}$.

Fig. 12 gives the syntax, big-step semantics and typing rules of MiniFJ$^{\vee}$. The subtyping relation $<:$ is the reflexive and transitive closure of the union of the extends relation and the standard rules for union:

$$\frac{}{T_1 <: T_1 \vee T_2} \qquad \frac{}{T_2 <: T_1 \vee T_2} \qquad \frac{T_1 <: T \quad T_2 <: T}{T_1 \vee T_2 <: T}$$

The functions mtype, fields and mbody are defined as for MiniFJ&λ, apart that here fields, method parameters and return types can be union types as well, still assuming the conditions on the class table (FJ1), (FJ2), and (FJ3).

Clearly rule (T-∨-ELIM) is inspired by rule (∨E), but restricted only to some specific contexts, named *(union) elimination contexts*. Elimination contexts are field access and method invocation, where the latter has $n > 0$ holes corresponding to the receiver and (for simplicity the first) $n - 1$ parameters. Thanks to this restriction, we are able to prove a standard inversion lemma, which is not known for the general rule in the λ-calculus.

Given an elimination context E, we denote by $E[e]$ the expression obtained by filling all holes of E by e.

This rule allows us to make the type system more "structural", with respect to FJ, similarly to what happens in FJ∨. Let us consider the following classes:

```
class C {
  A f; Object g;
  C update(A x) {...}
  Bool eq(C x) {..}
}
class D {
  A f;
  D update(A x) {...}
  Bool eq(D x) {...}
}
```

$$
\begin{array}{lcll}
e & ::= & x \mid e.\mathsf{f} \mid \mathsf{new}\ \mathsf{C}(e_1, \ldots, e_n) \mid e.\mathsf{m}(e_1, \ldots, e_n) & \text{expression} \\
 & & \mathtt{if}\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \mid \mathtt{true} \mid \mathtt{false} & \\
v & ::= & \mathsf{new}\ \mathsf{C}(v_1, \ldots, v_n) \mid \mathtt{true} \mid \mathtt{false} & \text{value} \\
T & ::= & \mathsf{C} \mid \mathtt{Bool} \mid T_1 \vee T_2 & \text{type} \\
E & ::= & [\,].\mathsf{f} \mid [\,].\mathsf{m}([\,], \ldots, [\,], e_1, \ldots, e_n) & \text{elimination context}
\end{array}
$$

$$
(\textsc{field})\ \frac{e \Rightarrow \mathsf{new}\ \mathsf{C}(v_1, \ldots, v_n)}{e.\mathsf{f}_i \Rightarrow v_i}\quad \begin{array}{l}\mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1; \ldots T_n\,\mathsf{f}_n; \\ i \in 1..n\end{array}
$$

$$
(\textsc{new})\ \frac{e_i \Rightarrow v_i \quad \forall i \in 1..n}{\mathsf{new}\ \mathsf{C}(e_1, \ldots, e_n) \Rightarrow \mathsf{new}\ \mathsf{C}(v_1, \ldots, v_n)}
$$

$$
(\textsc{invk})\ \frac{\begin{array}{l}e_0 \Rightarrow \mathsf{new}\ \mathsf{C}(vs') \\ e_i \Rightarrow v_i \quad \forall i \in 1..n \\ e[v_1/x_1] \ldots [v_n/x_n][\mathsf{new}\ \mathsf{C}(vs')/\mathtt{this}] \Rightarrow v\end{array}}{e_0.\mathsf{m}(e_1, \ldots, e_n) \Rightarrow v}\quad \mathsf{mbody}(\mathsf{C}, \mathsf{m}) = \langle x_1 \ldots x_n, e \rangle
$$

$$
(\textsc{true})\ \frac{}{\mathtt{true} \Rightarrow \mathtt{true}}\qquad (\textsc{false})\ \frac{}{\mathtt{false} \Rightarrow \mathtt{false}}
$$

$$
(\textsc{if-t})\ \frac{e \Rightarrow \mathtt{true} \quad e_1 \Rightarrow v}{\mathtt{if}\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \Rightarrow v}\qquad (\textsc{if-f})\ \frac{e \Rightarrow \mathtt{false} \quad e_2 \Rightarrow v}{\mathtt{if}\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \Rightarrow v}
$$

$$
(\textsc{t-var})\ \frac{}{\Gamma \vdash x : T}\quad \Gamma(x) = T\qquad (\textsc{t-bool})\ \frac{}{\Gamma \vdash b : \mathtt{Bool}}\quad b \in \{\mathtt{true}, \mathtt{false}\}
$$

$$
(\textsc{t-fld})\ \frac{\Gamma \vdash e : \mathsf{C}}{\Gamma \vdash e.\mathsf{f}_i : T_i}\quad \begin{array}{l}\mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1; \ldots T_n\,\mathsf{f}_n; \\ i \in 1..n\end{array}
$$

$$
(\textsc{t-new})\ \frac{\Gamma \vdash e_i : T_i \quad \forall i \in 1..n}{\Gamma \vdash \mathsf{new}\ \mathsf{C}(e_1, \ldots, e_n) : \mathsf{C}}\quad \mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1; \ldots T_n\,\mathsf{f}_n;
$$

$$
(\textsc{t-invk})\ \frac{\Gamma \vdash e : \mathsf{C} \quad \Gamma \vdash e_i : T_i \quad \forall i \in 1..n}{\Gamma \vdash e.\mathsf{m}(e_1, \ldots, e_n) : T}\quad \mathsf{mtype}(\mathsf{C}, \mathsf{m}) = T_1 \ldots T_n \to T
$$

$$
(\textsc{t-if})\ \frac{\Gamma \vdash e : \mathtt{Bool} \quad \Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathtt{if}\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : T}\qquad (\textsc{t-sub})\ \frac{\Gamma \vdash e : T}{\Gamma \vdash e : T'}\quad T <: T'
$$

$$
(\textsc{t-}\vee\textsc{-elim})\ \frac{\Gamma \vdash e : \bigvee_{i \in 1..m} \mathsf{C}_i \quad \Gamma, x{:}\mathsf{C}_i \vdash E[x] : T \quad \forall i \in 1..m}{\Gamma \vdash E[e] : T}\quad x \text{ fresh}
$$

Fig. 12. MiniFJ$^{\vee}$: syntax, big-step semantics and type system

They share a common structure, but they are not related by inheritance (there is no common superclass abstracting shared features), hence in standard FJ they cannot be handled uniformly. By means of (T-∨-ELIM) this is possible: for instance, we can write a wrapper class that, in a sense, provides the common interface of C and D "ex-post"

```
class CorD {
  C ∨ D el;
  A getf() { this.el.f }
  CorD update(A x) { new CorD(this.el.update(x)) }
}
```

Bodies of methods `getf` and `update` in class `CorD` are well-typed thanks to rule (T-∨-ELIM), as shown by the following derivation for `update`, where $\Gamma = x{:}\mathsf{A}, \mathtt{this}{:}\mathsf{CorD}$.

$$\dfrac{\dfrac{\Gamma \vdash \mathtt{this.el} : \mathsf{C} \vee \mathsf{D} \quad \dfrac{\Gamma, y{:}\mathsf{C} \vdash y.\mathtt{update}(x) : \mathsf{C}}{\Gamma, y{:}\mathsf{C} \vdash y.\mathtt{update}(x) : \mathsf{C} \vee \mathsf{D}} \quad \dfrac{\Gamma, y{:}\mathsf{D} \vdash y.\mathtt{update}(x) : \mathsf{D}}{\Gamma, y{:}\mathsf{D} \vdash y.\mathtt{update}(x) : \mathsf{C} \vee \mathsf{D}}}{\Gamma \vdash \mathtt{this.el.update}(x) : \mathsf{C} \vee \mathsf{D}}}{\Gamma \vdash \mathtt{new\ CorD(this.el.update}(x)\mathtt{)} : \mathsf{CorD}}$$

The above example can be typed in FJ∨ as well, even though with a different technique.[9] On the other hand, with our more uniform approach inspired by rule (VE), we can type examples where the same subexpression having a union type occurs more than once, and soundness relies on the determinism of evaluation, as in the example at the beginning of this section.

To illustrate this, let us consider an example. Assuming the above class table, consider the expression $e = \mathtt{if\ false\ then\ new\ C(\ldots)\ else\ ne}$ By rule (T-IF), the expression e has type $\mathsf{C} \vee \mathsf{D}$, and, by rule (T-∨-ELIM), the expression $e.\mathtt{eq}(e)$ has type `Bool`, as shown by the following derivation:

$$\dfrac{\vdash e : \mathsf{C} \vee \mathsf{D} \qquad x{:}\mathsf{C} \vdash x.\mathtt{eq}(x) : \mathtt{Bool} \quad x{:}\mathsf{D} \vdash x.\mathtt{eq}(x) : \mathtt{Bool}}{\vdash e.\mathtt{eq}(e) : \mathtt{Bool}}$$

This expression cannot be typed in FJ∨, because there is no way to eliminate the union type assigned to e when it occurs as an argument.

Quite surprisingly, subject reduction fails for the expected small-step semantics, even if there are no intersection types, which are the source, together with the (VE) rules, of the problems in the λ-calculus. Indeed, we have the following small-step reduction:

$$e.\mathtt{eq}(e) \longrightarrow \mathtt{new\ D(\ldots).eq}(e) \longrightarrow \mathtt{new\ D(\ldots).eq(new\ D(\ldots))}$$

where the intermediate expression cannot be typed, because e has a union type. This happens because intersection types are in a sense hidden in the class table: the method eq occurs in two different classes with different types, hence, roughly, we could assign it the intersection type $(\mathsf{C}\,\mathsf{C} \to \mathtt{Bool}) \wedge (\mathsf{D}\,\mathsf{D} \to \mathtt{Bool})$.

As in previous examples, the soundness proof uses an inversion lemma and a substitution lemma. The canonical forms lemma is trivial since the only values of type C are objects (constructor calls with values as arguments) instances of a subclass. In addition, we need a lemma (dubbed "key") which assures that a value typed by a union of classes can also be typed by one of these classes. The proof of this lemma is straightforward, since values having class types are just new constructors, as shown by canonical forms.

Lemma 8.12 (Substitution). *If* $\Gamma\{T'/x\} \vdash e : T$ *and* $\Gamma \vdash e' : T'$, *then* $\Gamma \vdash e[e'/x] : T'$.

Lemma 8.13 (Canonical forms). *The following hold:*

(1) If $\Gamma \vdash v : \mathtt{Bool}$, *then* $v = \mathtt{true}$ *or* $v = \mathtt{false}$.

(2) If $\Gamma \vdash v : \mathsf{C}$, *then* $v = \mathtt{new\ D}(v_1, \ldots, v_n)$ *and* $\mathsf{D} <: \mathsf{C}$.

Lemma 8.14 (Inversion). *The following hold:*

(1) If $\Gamma \vdash x : T$, *then* $\Gamma(x) <: T$.

[9] When the receiver of a method call has a union type, look-up (function mtype) is directly performed and gives a set of method signatures; arguments should comply all parameter types and the type of the call is the union of return types.

(2) *If* $\Gamma \vdash e.\mathsf{f} : T$, *then* $\Gamma \vdash e : \bigvee_{i \in 1..m} \mathsf{C}_i$ *and, for all* $i \in 1..m$, $\mathsf{fields}(\mathsf{C}_i) = T_{i1}\,\mathsf{f}_{i1}\,; \ldots T_{in_i}\,\mathsf{f}_{in_i}\,;$ *and* $\mathsf{f} = \mathsf{f}_{ik_i}$ *and* $T_{ik_i} <: T$ *for some* $k_i \in 1..n_i$.

(3) *If* $\Gamma \vdash \mathsf{new}\,\mathsf{C}(e_1, \ldots, e_n) : T$, *then* $\mathsf{C} <: T$ *and* $\mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1\,; \ldots T_n\,\mathsf{f}_n\,;$ *and* $\Gamma \vdash e_i : T_i$ *for all* $i \in 1..n$.

(4) *If* $\Gamma \vdash e_0.\mathsf{m}(e_1, \ldots, e_n) : T$, *then* $\Gamma \vdash e_0 : \bigvee_{i \in 1..m} \mathsf{C}_i$ *and, there is* $p \in 0..n$ *such that* $e_0 = \ldots = e_p$ *and, for all* $i \in 1..m$,

- $\mathsf{mtype}(\mathsf{C}_i, \mathsf{m}) = T_{i1} \ldots T_{in} \to T_i$, *and*
- *for all* $k \in 1..p$, $\mathsf{C}_i <: T_{ik}$, *and*
- *for all* $k \in p+1..n$, $\Gamma \vdash e_k : T_{ik}$, *and*
- $T_i <: T$.

(5) *If* $\Gamma \vdash \mathtt{if}\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : T$, *then* $\Gamma \vdash e : \mathtt{Bool}$ *and* $\Gamma \vdash e_i : T'$ *with* $T' <: T$ *and* $i \in 1..2$.

Proof Sketch. We prove only points 2 and 4.

(2) The proof is by induction on the derivation of $\Gamma \vdash e.\mathsf{f} : T$. For rule (T-FLD), we have $\Gamma \vdash e : \mathsf{C}$, $\mathsf{fields}(\mathsf{C}) = T_1\,\mathsf{f}_1\,; \ldots T_n\,\mathsf{f}_n\,;$, $\mathsf{f}_i = \mathsf{f}$ and $T_i = T$, for some $i \in 1..n$. For rule (T-SUB), the thesis is immediate by induction hypothesis. For rule (T-V-ELIM), we have $E = [\,].\mathsf{f}$, $\Gamma \vdash e : \bigvee_{i \in 1..m} \mathsf{C}_i$ and $\Gamma, x{:}\mathsf{C}_i \vdash E[x] : T$, for all $i \in 1..m$, then, by induction hypothesis, for all $i \in 1..m$, we get $\Gamma, x{:}\mathsf{C}_i \vdash x : \bigvee_{j \in 1..m_i} \mathsf{D}_{ij}$ and, for all $j \in 1..m_i$, $\mathsf{fields}(\mathsf{D}_{ij}) = T_{j11}\,\mathsf{f}_{j1}\,; \ldots T_{jn_j}\,\mathsf{f}_{jn_j}\,;$ and $T_{jk_j} <: T$, for some $k_j \in 1..n_j$. Since $\Gamma, x{:}\mathsf{C}_i \vdash x : \bigvee_{j \in 1..m_j} \mathsf{D}_{ij}$, we have $\mathsf{C}_i <: \bigvee_{j \in 1..m_j} \mathsf{D}_{ij}$, hence $\mathsf{C}_i <: \mathsf{D}_{ij_i}$, for some $j_i \in 1..m_i$, by definition of subtyping. Then the thesis follows easily by assumption (FJ2).

(4) The proof is by induction on the derivation of $\Gamma \vdash e_0.\mathsf{m}(e_1, \ldots, e_n) : T$. For rule (T-INVK), we have $\Gamma \vdash e_0 : \mathsf{C}_0$, $p = 0$, $\mathsf{mtype}(\mathsf{C}_0, \mathsf{m}) = T_1 \ldots T_n \to T$, and, for all $k \in 1..n$, $\Gamma \vdash e_k : T_k$. For rule (T-SUB), the thesis is immediate by induction hypothesis. For rule (T-V-ELIM), we have $E = [\,].\mathsf{m}([\,], \ldots, [\,], e_{p+1}, \ldots, e_n)$, hence p is the number of holes in E and $e_0 = \ldots = e_p$, and $\Gamma \vdash e_0 : \bigvee_{i \in 1..m} \mathsf{C}_i$ and, for all $i \in 1..m$, $\Gamma, x{:}\mathsf{C}_i \vdash E[x] : T$, with x fresh. By induction hypothesis, we know that, for all $i \in 1..m$, $\Gamma, x{:}\mathsf{C}_i \vdash x : \bigvee_{j \in 1..m_i} \mathsf{D}_{ij}$ and there is $p_i \in 1..n$ such that the first p_i arguments of $E[x]$ are equal to the receiver, namely x and this implies $p_i \le p$ because x is fresh. Let $i \in 1..m$. Since $\Gamma, x{:}\mathsf{C}_i \vdash x : \bigvee_{j \in 1..m_i} \mathsf{D}_{ij}$, we get $\mathsf{C}_i <: \bigvee_{j \in 1..m_j} \mathsf{D}_{ij}$, thus $\mathsf{C}_i <: \mathsf{D}_{ij_i}$, for some $j_i \in 1..m_i$, by definition of subtyping. Therefore, by induction hypothesis and assumption (FJ3), we get $\mathsf{mtype}(\mathsf{C}_i, \mathsf{m}) = T_{i1} \ldots T_{in} \to T_i$ and, for all $k \in 1..p_i$, $\mathsf{D}_{ij_i} <: T_{ik}$, hence $\mathsf{C}_i <: T_{ik}$, and, for all $k \in p_i + 1..p$, $\Gamma, x{:}\mathsf{C}_i \vdash x : T_{ik}$, hence $\mathsf{C}_i <: T_{ik}$ and, for all $k \in p+1..n$, $\Gamma, x{:}\mathsf{C}_i \vdash e_k : T_{ik}$, hence, because x does not occur in e_k as it is fresh, by contraction we get $\Gamma \vdash e_k : T_{ik}$, and, finally, $T_i <: T$.

□

Lemma 8.15 (Key). *If* $\Gamma \vdash v : \bigvee_{1 \le i \le n} \mathsf{C}_i$, *then* $\Gamma \vdash v : \mathsf{C}_i$ *for some* $i \in 1 \ldots n$.

In order to state soundness, let $\langle C_4, R_4, \mathcal{R}_4 \rangle$ be the big-step semantics defined in Fig. 12 (C_4 is the set of expressions and R_4 is the set of values), and let $\Pi 4^C_T = \{e \in C_4 \mid \vdash e : T\}$ and $\Pi 4^R_T = \{v \in R_4 \mid \vdash v : T\}$, for T defined in Fig. 12. We need a last lemma to prove soundness:

Lemma 8.16 (Determinism). *If* $\mathcal{R}_4 \vdash_\mu e \Rightarrow v_1$ *and* $\mathcal{R}_4 \vdash_\mu e \Rightarrow v_2$, *then* $v_1 = v_2$.

Proof. Straightforward induction on rules in $\mathcal{R}_4$, because every syntactic construct has a unique big-step meta-rule. □

$$
\begin{array}{lcll}
e & ::= & x \mid e.\mathsf{f} \mid \mathsf{new}\ \mathsf{C}(e_1,\ldots,e_n) \mid e.\mathsf{m}(e_1,\ldots,e_n) \mid e.\mathsf{f}{=}e' \mid \iota & \text{expressions} \\
c & ::= & \langle \mathcal{M}, e\rangle & \text{configurations} \\
r & ::= & \langle \mathcal{M}, \iota\rangle & \text{results}
\end{array}
$$

$$
\textsc{(obj)}\ \frac{}{\langle \mathcal{M}, \iota\rangle \Rightarrow \langle \mathcal{M}, \iota\rangle}
\qquad
\textsc{(fld)}\ \frac{\langle \mathcal{M}, e\rangle \Rightarrow \langle \mathcal{M}', \iota\rangle}{\langle \mathcal{M}, e.\mathsf{f}_i\rangle \Rightarrow \langle \mathcal{M}', \iota_i\rangle}\quad \begin{array}{l}\mathcal{M}'(\iota) = \mathsf{new}\ \mathsf{C}(\iota_1,\ldots,\iota_n)\\ \mathsf{fields}(\mathsf{C}) = \mathsf{C}_1\,\mathsf{f}_1;\ldots\mathsf{C}_n\,\mathsf{f}_n;\\ i \in 1..n\end{array}
$$

$$
\textsc{(new)}\ \frac{\langle \mathcal{M}_i, e_i\rangle \Rightarrow \langle \mathcal{M}_{i+1}, \iota_i\rangle \quad \forall i \in 1..n}{\langle \mathcal{M}, \mathsf{new}\ \mathsf{C}(e_1,\ldots,e_n)\rangle \Rightarrow \langle \mathcal{M}', \iota\rangle}\quad \begin{array}{l}\mathcal{M}_1 = \mathcal{M}\\ \mathcal{M}' = \mathcal{M}_{n+1}\{\mathsf{new}\ \mathsf{C}(\iota_1,\ldots,\iota_n)/\iota\}\\ \iota \text{ fresh}\end{array}
$$

$$
\textsc{(invk)}\ \frac{\begin{array}{l}\langle \mathcal{M}_i, e_i\rangle \Rightarrow \langle \mathcal{M}_{i+1}, \iota_i\rangle \quad \forall i \in 0..n\\ \langle \mathcal{M}_{n+1}, e[\iota_1/x_1]\ldots[\iota_n/x_n][\iota_0/\mathtt{this}]\rangle \Rightarrow \langle \mathcal{M}', \iota\rangle\end{array}}{\langle \mathcal{M}, e_0.\mathsf{m}(e_1,\ldots,e_n)\rangle \Rightarrow \langle \mathcal{M}', \iota\rangle}\quad \begin{array}{l}\mathcal{M}_0 = \mathcal{M}\\ \mathcal{M}_1(\iota_0) = \mathsf{new}\ \mathsf{C}(\_)\\ \mathsf{mbody}(\mathsf{C},\mathsf{m}) = \langle x_1\ldots x_n, e\rangle\end{array}
$$

$$
\textsc{(fld-up)}\ \frac{\langle \mathcal{M}, e\rangle \Rightarrow \langle \mathcal{M}', \iota\rangle \quad \langle \mathcal{M}', e'\rangle \Rightarrow \langle \mathcal{M}'', \iota'\rangle}{\langle \mathcal{M}, e.\mathsf{f}_i{=}e'\rangle \Rightarrow \langle \mathcal{M}''_{[\iota.i=\iota']}, \iota'\rangle}\quad \begin{array}{l}\mathcal{M}(\iota) = \mathsf{new}\ \mathsf{C}(\iota_1,\ldots,\iota_n)\\ \mathsf{fields}(\mathsf{C}) = \mathsf{C}_1\,\mathsf{f}_1;\ldots\mathsf{C}_n\,\mathsf{f}_n;\\ i \in 1..n\end{array}
$$

Fig. 13. Imperative FJ: syntax and big-step semantics

Theorem 8.17 (Soundness). *The big-step semantics $\langle C_4, R_4, \mathcal{R}_4\rangle$ and the indexed predicate* $\Pi 4$ *satisfy the conditions (lp), (∃p) and (∀p) of Section 7.2.*

Proof Sketch. We sketch the proof only of (lp) for rule (invk), other cases and conditions are similar to previous proofs.

For rule (invk), Lemma 8.14 (4) applied to $\vdash e_0.\mathsf{m}(e_1,\ldots,e_n) : T$ implies $\vdash e_0 : \bigvee_{i\in 1..m} \mathsf{C}_i$ and, there is $p \in 0..n$ such that $e_0 = \ldots = e_p$ and, for all $i \in 1..m$, $\mathsf{mtype}(\mathsf{C}_i,\mathsf{m}) = T_{i1}\ldots T_{in} \to T_i$, and for all $k \in 1..p$, $\mathsf{C}_i <: T_{ik}$, and for all $k \in p+1..n$, $\vdash e_i : T_{ik}$, and $T_i <: T$. Assuming $\vdash \mathsf{new}\ \mathsf{C}(vs) : \bigvee_{i\in 1..m} \mathsf{C}_i$, by Lemma 8.15 and Lemma 8.13, we get $\mathsf{C} <: \mathsf{C}_i$ for some $i \in 1..m$. Since $\mathsf{mtype}(\mathsf{C}_i,\mathsf{m}) = T_{i1}\ldots T_{in} \to T_i$ and $\mathsf{mbody}(\mathsf{C},\mathsf{m}) = \langle x_1\ldots x_n, e\rangle$, by assumption (FJ3) and (FJ1), $\mathtt{this}{:}\mathsf{C}, x_1{:}T_{i1}, \ldots, x_n{:}T_{in} \vdash e : T_i$. Assume, for all $k \in 1..p$, $\vdash v_k : \bigvee_{i\in 1..m} \mathsf{C}_i$ and, for all $k \in p+1..n$, $\vdash v_k : T_{ik}$, then, since $e_0 = \ldots = e_p$, by Lemma 8.16, we get $v_1 = \ldots = v_p = \mathsf{new}\ \mathsf{C}(vs)$, hence $\vdash v_k : T_{ik}$, for all $k \in 1..p$, because $\mathsf{C} <: \mathsf{C}_i <: T_{ik}$ for all $k \in 1..p$. Lemma 8.12 gives $\vdash e[v_1/x_1]\ldots[v_n/x_n][\mathsf{new}\ \mathsf{C}(vs)/\mathtt{this}] : T_i$. Finally, we can conclude $\vdash v : T$ by rule (t-sub), as $T_i <: T$. □

8.5 Imperative FJ

We show here how our technique behaves in an imperative setting. In Fig. 13 and Fig. 14 we show a minimal imperative extension of FJ. We assume a well-typed class table and we use the notations introduced in Section 8.2. Expressions are enriched with field assignment and *object identifiers* ι, which only occur in runtime expressions. A *memory* $\mathcal{M}$ maps object identifiers to *object states*, which are expressions of shape $\mathsf{new}\ \mathsf{C}(\iota_1,\ldots\iota_n)$. Results are configurations of shape $\langle \mathcal{M}, \iota\rangle$. We denote by $\mathcal{M}_{[\iota.i=\iota']}$ the memory obtained from $\mathcal{M}$ by replacing by ι' the i-th field of the object state associated with ι. The *type assignment* Σ maps object identifiers into types (class names). We write $\Sigma \vdash e : \mathsf{C}$ for $\emptyset; \Sigma \vdash e : \mathsf{C}$.

As for the other examples, to prove soundness we need some standard properties of the typing rules: inversion and substitution lemmas.

$$\text{(T-CONF)}\ \frac{\Sigma \vdash \mathcal{M}(\iota) : \Sigma(\iota)\ \forall \iota \in \mathrm{dom}(\mathcal{M}) \quad \Sigma \vdash e : \mathsf{C}}{\Sigma \vdash \langle \mathcal{M}, e\rangle : \mathsf{C}} \quad \mathrm{dom}(\Sigma) = \mathrm{dom}(\mathcal{M})$$

$$\text{(T-VAR)}\ \frac{}{\Gamma; \Sigma \vdash x : \mathsf{C}} \quad \Gamma(x) = \mathsf{C}$$

$$\text{(T-FLD)}\ \frac{\Gamma; \Sigma \vdash e : \mathsf{C}}{\Gamma; \Sigma \vdash e.\mathsf{f}_i : \mathsf{C}_i} \quad \begin{array}{l}\mathsf{fields}(\mathsf{C}) = \mathsf{C}_1\,\mathsf{f}_1; \ldots \mathsf{C}_n\,\mathsf{f}_n; \\ i \in 1..n\end{array}$$

$$\text{(T-NEW)}\ \frac{\Gamma; \Sigma \vdash e_i : \mathsf{C}_i \quad \forall i \in 1..n}{\Gamma; \Sigma \vdash \mathsf{new}\,\mathsf{C}(e_1, \ldots, e_n) : \mathsf{C}} \quad \mathsf{fields}(\mathsf{C}) = \mathsf{C}_1\,\mathsf{f}_1; \ldots \mathsf{C}_n\,\mathsf{f}_n;$$

$$\text{(T-INVK)}\ \frac{\Gamma; \Sigma \vdash e_i : \mathsf{C}_i \quad \forall i \in 0..n}{\Gamma; \Sigma \vdash e_0.\mathsf{m}(e_1, \ldots, e_n) : \mathsf{C}} \quad \mathsf{mtype}(\mathsf{C}_0, \mathsf{m}) = \mathsf{C}_1 \ldots \mathsf{C}_n \to \mathsf{C}$$

$$\text{(T-FLD-UP)}\ \frac{\begin{array}{c}\Gamma; \Sigma \vdash e : \mathsf{C} \\ \Gamma; \Sigma \vdash e' : \mathsf{C}_i\end{array}}{\Gamma; \Sigma \vdash e.\mathsf{f}_i{=}e' : \mathsf{C}_i} \quad \begin{array}{l}\mathsf{fields}(\mathsf{C}) = \mathsf{C}_1\,\mathsf{f}_1; \ldots \mathsf{C}_n\,\mathsf{f}_n; \\ i \in 1..n\end{array}$$

$$\text{(T-OID)}\ \frac{}{\Gamma; \Sigma \vdash \iota : \mathsf{C}} \quad \Sigma(\iota) = \mathsf{C} \qquad \text{(T-SUB)}\ \frac{\Gamma; \Sigma \vdash e : \mathsf{C}}{\Gamma; \Sigma \vdash e : \mathsf{C}'} \quad \mathsf{C} <: \mathsf{C}'$$

Fig. 14. Imperative FJ: typing rules

Lemma 8.18 (Inversion). *The following hold:*

(1) If $\Sigma \vdash \langle \mathcal{M}, e\rangle : \mathsf{C}$, *then* $\Sigma \vdash \mathcal{M}(\iota) : \Sigma(\iota)$ *for all* $\iota \in dom(\mathcal{M})$ *and* $\Sigma \vdash e : \mathsf{C}$ *and* $dom(\Sigma) = dom(\mathcal{M})$.

(2) If $\Gamma; \Sigma \vdash x : \mathsf{C}$, *then* $\Gamma(x) <: \mathsf{C}$.

(3) If $\Gamma; \Sigma \vdash e.\mathsf{f}_i : \mathsf{C}$, *then* $\Gamma; \Sigma \vdash e : \mathsf{D}$ *and* $\mathsf{fields}(\mathsf{D}) = \mathsf{C}_1\,\mathsf{f}_1; \ldots \mathsf{C}_n\,\mathsf{f}_n;$ *and* $\mathsf{C}_i <: \mathsf{C}$ *where* $i \in 1..n$.

(4) If $\Gamma; \Sigma \vdash \mathsf{new}\,\mathsf{C}(e_1, \ldots, e_n) : \mathsf{D}$, *then* $\mathsf{C} <: \mathsf{D}$ *and* $\mathsf{fields}(\mathsf{C}) = \mathsf{C}_1\,\mathsf{f}_1; \ldots \mathsf{C}_n\,\mathsf{f}_n;$ *and* $\Gamma; \Sigma \vdash e_i : \mathsf{C}_i$ *for all* $i \in 1..n$.

(5) If $\Gamma; \Sigma \vdash e_0.\mathsf{m}(e_1, \ldots, e_n) : \mathsf{C}$, *then* $\Gamma; \Sigma \vdash e_i : \mathsf{C}_i$ *for all* $i \in 0..n$ *and* $\mathsf{mtype}(\mathsf{C}_0, \mathsf{m}) = \mathsf{C}_1 \ldots \mathsf{C}_n \to \mathsf{D}$ *with* $\mathsf{D} <: \mathsf{C}$.

(6) If $\Gamma; \Sigma \vdash e.\mathsf{f}_i{=}e' : \mathsf{C}$, *then* $\Gamma; \Sigma \vdash e : \mathsf{D}$ *and* $\mathsf{fields}(\mathsf{D}) = \mathsf{C}_1\,\mathsf{f}_1; \ldots \mathsf{C}_n\,\mathsf{f}_n;$, *with* $i \in 1..n$, *and* $\Gamma; \Sigma \vdash e' : \mathsf{C}_i$ *and* $\mathsf{C}_i <: \mathsf{C}$.

(7) If $\Gamma; \Sigma \vdash \iota : \mathsf{C}$, *then* $\Sigma(\iota) <: \mathsf{C}$.

Lemma 8.19 (Substitution). *If* $\Gamma\{\mathsf{C}'/x\}; \Sigma \vdash e : \mathsf{C}$ *and* $\Gamma; \Sigma \vdash e' : \mathsf{C}'$, *then* $\Gamma; \Sigma \vdash e[e'/x] : \mathsf{C}$.

Let $\langle C_5, R_5, \mathcal{R}_5\rangle$ be the big-step semantics defined in Fig. 13. We can prove the soundness of the indexed predicate $\Pi 5$ defined by: $\Pi 5^{C}_{\langle \Sigma, \mathsf{C}\rangle} = \{\langle \mathcal{M}, e\rangle \in C_5 \mid \Sigma' \vdash \langle \mathcal{M}, e\rangle : \mathsf{C} \text{ for some } \Sigma' \text{ s.t. } \Sigma \subseteq \Sigma'\}$ and $\Pi 5^{R}_{\langle \Sigma, \mathsf{C}\rangle} = R_5 \cap \Pi 5^{C}_{\langle \Sigma, \mathsf{C}\rangle}$. The type assignment Σ' is needed, since memory can grow during evaluation.

Theorem 8.20 (Soundness). *The big-step semantics* $\langle C_5, R_5, \mathcal{R}_5\rangle$ *and the indexed predicate* $\Pi 5$ *satisfy the conditions (LP), (∃P) and (∀P) of Section 7.2.*

Proof. We prove separately the three conditions. The most interesting aspect here is that the presence of a memory induces a dependency between subsequent premises in each big-step rule and the hypotheses provided by the soundness conditions are essential to handle such a dependency.

Proof of (LP): The proof is by cases on instantiations of meta-rules.

Case: (OBJ) Trivial from the hypothesis.

Case: (FLD) Lemma 8.18 (1) applied to $\Sigma \vdash \langle \mathcal{M}, e.\mathsf{f}_i\rangle : \mathsf{C}$ implies $\Sigma \vdash \mathcal{M}(\iota) : \Sigma(\iota)$ for all $\iota \in \mathrm{dom}(\mathcal{M})$ and $\Sigma \vdash e.\mathsf{f}_i : \mathsf{C}$ and $\mathrm{dom}(\Sigma) = \mathrm{dom}(\mathcal{M})$. Lemma 8.18 (3) applied to $\Sigma \vdash e.\mathsf{f}_i : \mathsf{C}$ implies $\Sigma \vdash e : \mathsf{D}$ and $\mathsf{fields}(\mathsf{D}) = \mathsf{C}_1\,\mathsf{f}_1; \ldots \mathsf{C}_n\,\mathsf{f}_n;$ and

$C_i <: C$ where $i \in 1..n$. Since $\langle \mathcal{M}, e\rangle \Rightarrow \langle \mathcal{M}', \iota\rangle$ is a premise we assume $\Sigma' \vdash \langle \mathcal{M}', \iota\rangle : D$ with $\Sigma \subseteq \Sigma'$. Lemma 8.18 (1) and Lemma 8.18 (7) imply $\Sigma'(\iota) <: D$. Lemma 8.18 (4) allows us to get $\mathcal{M}'(\iota) = \mathsf{new}\, C'(\iota_1, \dots \iota_m)$ with $n \leq m$ and $C' <: D$ and $\Sigma' \vdash \iota_i : C_i$. So we conclude $\Sigma' \vdash \langle \mathcal{M}', \iota_i\rangle : C$ by rules (T-SUB) and (T-CONF).

Case: (NEW) Lemma 8.18 (1) applied to $\Sigma \vdash \langle \mathcal{M}, \mathsf{new}\, C(e_1, \dots, e_n)\rangle : D$ implies $\Sigma \vdash \mathcal{M}(\iota) : \Sigma(\iota)$ for all $\iota \in \mathsf{dom}(\mathcal{M})$ and $\Sigma \vdash \mathsf{new}\, C(e_1, \dots, e_n) : D$ and $\mathsf{dom}(\Sigma) = \mathsf{dom}(\mathcal{M})$. Lemma 8.18 (4) applied to $\Sigma \vdash \mathsf{new}\, C(e_1, \dots, e_n) : D$ implies $C <: D$ and $\mathsf{fields}(C) = C_1\, f_1\, ; \dots C_n\, f_n\, ;$ and $\Sigma \vdash e_i : C_i$ for all $i \in 1..n$. Since $\langle \mathcal{M}, e_i\rangle \Rightarrow \langle \mathcal{M}_{i+1}, \iota_i\rangle$ is a premise we assume $\Sigma_i \vdash \langle \mathcal{M}_{i+1}, \iota_i\rangle : C_i$ for all $i \in 1..n$ with $\Sigma \subseteq \Sigma_1 \subseteq \dots \subseteq \Sigma_n$. Lemma 8.18 (1) and Lemma 8.18 (7) imply $\Sigma_i(\iota_i) <: C_i$ for all $i \in 1..n$. Using rules (T-OID), (T-NEW) and (T-SUB) we derive $\Sigma_n \vdash \mathsf{new}\, C(\iota_1, \dots, \iota_n) : D$. We then conclude $\Sigma_n, \iota : D \vdash \langle \mathcal{M}_{n+1}, \iota\rangle : D$ by rules (T-OID) and (T-CONF).

Case: (INVK) Lemma 8.18 (1) applied to $\Sigma_0 \vdash \langle \mathcal{M}_0, e_0.\mathsf{m}(e_1, \dots, e_n)\rangle : C$ implies $\Sigma_0 \vdash \mathcal{M}_0(\iota) : \Sigma_0(\iota)$ for all $\iota \in \mathsf{dom}(\mathcal{M}_0)$ and $\Sigma_0 \vdash e_0.\mathsf{m}(e_1, \dots, e_n) : C$ and $\mathsf{dom}(\Sigma_0) = \mathsf{dom}(\mathcal{M}_0)$. Lemma 8.18 (5) applied to $\Sigma_0 \vdash e_0.\mathsf{m}(e_1, \dots, e_n) : C$ implies $\Sigma_i \vdash e_i : C_i$ for all $i \in 0..n$ and $\mathsf{mtype}(C_0, \mathsf{m}) = C_1 \dots C_n \to D$ with $D <: C$. Since $\langle \mathcal{M}_i, e_i\rangle \Rightarrow \langle \mathcal{M}_{i+1}, \iota_i\rangle$ is a premise we assume $\Sigma_i \vdash \langle \mathcal{M}_{i+1}, \iota_i\rangle : C_i$ for all $i \in 0..n$ with $\Sigma_0 \subseteq \dots \subseteq \Sigma_n$. Lemma 8.18 (1) gives $\Sigma_i \vdash \iota_i : C_i$ for all $i \in 0..n$. The typing of the class table implies $x_1{:}C_1, \dots, x_n{:}C_n, \mathtt{this}{:}C_0 \vdash e : D$. Lemma 8.19 gives $\Sigma_n \vdash e' : D$ where $e' = e[\iota_1/x_1] \dots [\iota_n/x_n][\iota_0/\mathtt{this}]$. Using rules (T-SUB) and (T-CONF) we derive $\Sigma_n \vdash \langle \mathcal{M}_{n+1}, e'\rangle : C$. Since $\langle \mathcal{M}_{n+1}, e'\rangle \Rightarrow \langle \mathcal{M}', \iota\rangle$ is a premise we conclude $\Sigma' \vdash \langle \mathcal{M}', \iota\rangle : C$ with $\Sigma_n \subseteq \Sigma'$.

Case: (FLD-UP) Lemma 8.18 (1) applied to $\Sigma \vdash \langle \mathcal{M}, e.\mathsf{f}_i{=}e'\rangle : C$ implies $\Sigma \vdash \mathcal{M}(\iota) : \Sigma(\iota)$ for all $\iota \in \mathsf{dom}(\mathcal{M})$ and $\Sigma \vdash e.\mathsf{f}_i{=}e' : C$ and $\mathsf{dom}(\Sigma) = \mathsf{dom}(\mathcal{M})$. Lemma 8.18 (6) applied to $\Sigma \vdash e.\mathsf{f}_i{=}e' : C$ implies $\Sigma \vdash e : D$ and $\mathsf{fields}(D) = C_1\, f_1\, ; \dots C_n\, f_n\, ;$ and $\Sigma \vdash e' : C_i$ and $C_i <: C$. Since $\langle \mathcal{M}, e\rangle \Rightarrow \langle \mathcal{M}', \iota\rangle$ and $\langle \mathcal{M}', e'\rangle \Rightarrow \langle \mathcal{M}'', \iota'\rangle$ are premises we assume $\Sigma' \vdash \langle \mathcal{M}', \iota\rangle : D$ and $\Sigma'' \vdash \langle \mathcal{M}'', \iota'\rangle : C_i$, with $\Sigma \subseteq \Sigma' \subseteq \Sigma''$. Notice that $\mathcal{M}''(\iota)$ and $\mathcal{M}''_{[\iota.i=\iota']}(\iota)$ have the same types for all ι by construction. We conclude $\Sigma'' \vdash \langle \mathcal{M}''_{[\iota.i=\iota']}, \iota'\rangle : C_i$.

Proof of (∃P): All the closed expressions appear as conclusions in the reduction rules.

Proof of (∀P): Since the only values are configurations with object identifiers it is easy to verify that the premises of the reduction rules are satisfied, being the conditions on memory and object identifiers assured by the typing rules. □

9 CONCLUDING DISCUSSIONS

The big-step style can be useful for abstracting details or directly deriving the implementation of an interpreter. However, reasoning on properties involving infinite computations, such as the soundness of a type system, is non-trivial, because standard big-step semantics is able only to capture finite computations, hence it cannot distinguish between stuck and infinite ones.

In this paper, we address this problem, providing a systematic analysis of big-step semantics. The first, and fundamental, methodological feature of our analysis is that we want to be *independent from specific languages*, developing an abstract study of big-step semantics in itself. Therefore, we provide a definition of what a big-step semantics is, so our results will be applicable, as we show by several examples, to all concrete big-step semantics matching our definition.

A second important building block of our approach is that we take seriously the fact that big-step rules implicitly define an *evaluation algorithm*. Indeed, we make such intuition formal by showing that starting from the rules we can define a transition relation on incomplete derivations, abstractly modeling such evaluation algorithm. Relying on this transition relation, we are able to define computations in the big-step semantics in the usual way, as possibly infinite sequences of transition steps; thus we can distinguish converging, diverging and stuck computations, even though

big-step rules only define convergence. This shows that diverging and stuck computations are, in a sense, implicit in standard big-step rules, and the transition relation makes them explicit.

Finally, the third feature of our approach is that we provide *constructions* that, starting from a usual big-step semantics, produce an extended one where the distinction between diverging and stuck computation is explicit. Such constructions show that we can distinguish stuckness and divergence directly by a big-step semantics, without resorting to a transition relation: we rely on the above described transition relation on incomplete derivations only to prove that the constructions are correct. Corules are crucial to define extended big-step semantics precisely modelling divergence just as a special result, thus avoiding the redundancy introduced by traces.

Building on this systematic study, we show how one can reason about soundness of a predicate directly on a big-step semantics. To this end, we design proof techniques for two flavours of soundness, based on sufficient conditions on big-step rules.

9.1 Related work

The research presented in this paper follows a stream of work dating back to Cousot and Cousot [21], who proposed a stratified approach, investigated by Leroy and Grall [36] as well, with a separate judgment for divergence, defined coinductively. In this way, however, there is no unique formal definition of the behaviour of the modelled system. An alternative possibility, also investigated by Leroy and Grall [36], is to interpret coinductively the standard big-step rules (*coevaluation*). Unfortunately, coevaluation is non-deterministic, allowing the derivation of spurious judgements, and, thus, may fail to correctly capture the infinite behavior of a configuration: a diverging term, such as Ω, evaluates to any value, hence it cannot be properly distinguished from converging terms. Furthermore, in coevaluation there are still configurations, such as $\Omega\,(0\,0)$, for which no judgment can be derived, here because no judgment can be derived for the subterm $0\,0$; basically, this is due to the fact that divergence of a premise should be propagated and this cannot be correctly handled by coevaluation as divergence is not explicitly modelled.

Pretty big-step semantics by Charguéraud [18] handles the issue of duplication of meta-rules by a unified judgment with a unique set of (meta-)rules and divergence modelled by a special value. Rules are interpreted coinductively, hence they allow the derivation of spurious judgements, but, thanks to the use of a special value for divergence and the particular structure of rules, they can solve most of the issues of coevaluation. However, this particular structure of rules is not as natural as usual big-step rules and, more importantly, it requires the introduction of new specific syntactic forms representing intermediate computation steps, as in small-step semantics, hence making the big-step semantics less abstract. This may be a problem, for instance, when proving soundness of a type system, as such intermediate configurations may be ill-typed.

Poulsen and Mosses [48] subsequently present *flag-based big-step semantics*, which further streamlines the approach by combining it with the M-SOS technique (modular structural operational semantics), thereby reducing the number of (meta-)rules and premises, avoiding the need for intermediate configurations. The key idea is to extend configurations and results by flags explicitly modelling convergence and divergence, used to properly handle divergence propagation. To model divergence, they interpret rules coinductively, hence they allow the derivation of spurious judgements.

Differently from all the previously cited papers, which consider specific examples, the work by Ager [4] shares with us the aim of providing a generic construction to model non-termination, basing on an arbitrary big-step semantics. Ager considers a big-step judgement of shape $\rho \vdash t \Downarrow v$ where ρ is an environment, t a syntactic term and v a final value, and values, environments and the signature for terms are left unspecified. Then, given a big-step semantics, he describes a method to extract an abstract machine from it, which models a proof-search algorithm. In this way, converging,

diverging and stuck computations are distinguished. This approach is somehow similar to our transition relation on partial evaluation trees, even tough a different style is used: we have no syntactic components and the transition system we propose is directly defined on evaluation trees and corresponds to a partial order on them, modelling refinement. Moreover, Ager's notion of big-step semantics is not fully formal, in particular, it is not clear whether he works with plain rules or meta-rules.

Another piece of work whose aim is to define a general framework for operational semantics specification is the one by Bodin et al. [14] on *skeletal semantics*. Here the key idea is to specify the semantics of a language by a set of skeletons, one for each syntactic construct, which describe how to evaluate each of them. Skeletons are very much like big-step rules, indeed they can be regarded as an ad-hoc syntax for specifying them. This syntax is quite unusual, but probably better suited for the Coq implementation which the framework comes with. This approach is not specifically tailored for big-step semantics: a skeletal specification can give rise to semantics in different styles, such as big-step, small-step or abstract machines. However, given the similarity between skeletons and big-step rules, it may be possible to adapt the proof technique we propose to this setting, but this is matter for future work.

Ancona et al. [9] firstly show that with corules one can define a unified big-step judgment with a unique set of rules avoiding spurious evaluations. This can be seen as *constrained coevaluation*. Indeed, corules add constraints on the infinite derivations to filter out spurious results, so that, for diverging terms, it is only possible to get ∞ as result. This is extended to include observations as traces by Ancona et al. [10]. A further step is done by Ancona et al. [7], where observations are modelled by an arbitrary monoid and a variant of the construction described in Section 6 is considered.

Other proposals, by Amin and Rompf [5], Owens et al. [43], are inspired by *definitional interpreters* [49], based on a step-indexed approach (a.k.a."fuel"-based semantics) where computations are approximated to some finite amount of steps (typically with a counter); in this way divergence can be modeled by induction. Owens et al. [43] investigates functional big-step semantics for proving by induction compiler correctness. Amin and Rompf [5] explore inductive proof strategies for type soundness properties for the polymorphic type systems $F_{<:}$, and equivalence with small-step semantics. An inductive proof of type soundness for the big-step semantics of a Java-like language is proposed by Ancona [6].

Coinductive trace semantics in big-step style have been studied by Nakata and Uustalu [40, 41, 42]. Their investigation started with the semantics of an imperative While language with no I/O [40], 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 [41]. They provide a constructive theory and metatheory, together with a Coq formalization of their results. Differently from our approach, weak bisimilarity between traces is needed for proving that programs exhibit equivalent observable behaviors. This is due to the fact that "silent effects" (that is, non-observable internal steps) must be explicitly represented to guarantee guardedness conditions which ensure productivity of corecursive definitions. This is a natural consequence of having computable definitions. By using corules, we can avoid bisimilarity, accepting an approach which is not fully constructive.

This semantics has been subsequently extended with interactive I/O [42], by exploiting the notion of resumption monad: 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. A generalised notion of resumption has been introduced later by Piróg and Gibbons [45] in a category-theoretic and coalgebraic context.

Danielsson [28], inspired by Leroy and Grall [36], relying on the coinductive partiality monad, defines big-step semantics for λ-calculi and virtual machines as total, computable functions able to capture divergence.

The resumption monad of Nakata and Uustalu [42] and the partiality monad of Danielsson [28] are inspired by the seminal work of Capretta [15] on the *delay monad*, where coinductive types are exploited to model infinite computations by means of a type constructor for partial elements, which allows the formal definition of convergence and divergence and a type-theoretic representation of general recursive functions; this type constructor is proved to constitute a strong monad, upon which subsequent related papers [2, 17, 38] elaborated to define other monads for managing divergence. In particular, McBride [38] has proposed a more general approach based on a free monad for which the delay monad is an instantiation obtained through a monad morphism. All these proposals are based on the step-indexed approach.

More recently, interaction trees (ITrees) [54] have been presented as a coinductive variant of free monads with the main aim of defining the denotational semantics for effectful and possibly nonterminating computations, to allow compositional reasoning for mutually recursive components of an interactive system, with fully mechanized proofs in Coq. Interaction trees are coinductively defined trees which directly support a more general fixpoint combinator which does need a step-indexed approach, as happens for the general monad of McBride. A `Tau` constructor is introduced to represent a silent step of computation, to express silently diverging computations without violating Coq's guardedness condition; as a consequence, a generic definition of weak bisimulation on ITrees is required to remove any finite number of `Taus`, similarly as what happens in the approach of Nakata and Uustalu.

9.2 Future work

There are several directions for further research. A first direction is to study other approaches to model divergence in big-step semantics using our general meta-theory, that is, defining yet other constructions, such as adding a counter and timeout, as done by Amin and Rompf [5], Owens et al. [43], or adding flags, as done by Poulsen and Mosses [48]. This would provide a general account of these approaches, allowing to study their properties in general, abstracting away particular features of concrete languages. A further direction is to consider other computational models such as probabilistic computations, which are quite difficult to model in big-step style, as shown by Dal Lago and Zorzi [27].

Concerning proof techniques for soundness, we also plan to compare our proof technique with the standard one for small-step semantics: if a predicate satisfies progress and subject reduction with respect to a small-step semantics, does it satisfy our soundness conditions with respect to an equivalent big-step semantics? To formally prove such a statement, the first step will be to express equivalence between small-step and big-step semantics, and such equivalence has to be expressed at the level of big-step rules, as it needs to be extendible to stuck and infinite computations. Note that, as a by-product, this will provide us with a proof technique to show equivalence between small-step and big-step semantics. Ancona et al. [7] make a first attempt to express such an equivalence for a more restrictive class of big-step semantics. On the other hand, the converse does not hold, as shown by the examples in Section 8.2 and Section 8.4.

Furthermore, it would be interesting to extend such techniques for soundness to big-step semantics with observations, taking inspiration from type and effect systems [37, 52].

Last but not least, to support reasoning by our framework on concrete examples, such as those in Section 8, it is desirable to have a mechanisation of our meta-theory and related techniques. A necessary preliminary step in this direction is to provide support for corules in proof assistants. An Agda library supporting (generalised) inference systems is described by Ciccone et al. [19] and can be found at https://github.com/LcicC/inference-systems-agda. Moreover, in the paper we lazily relied on the usual setting of classical logic (even though we try not to abuse of it), however, towards a formalisation, we will have to carefully rearrange definitions and proofs to fit the logic of the choosen proof assistant.

ACKNOWLEDGMENTS

Special thanks go to Elena Zucca, Mariangiola Dezani-Ciancaglini and Viviana Bono for collaborating on this work with many interesting discussions and useful suggestions, which have greatly improved the paper.

REFERENCES

[1] Martín Abadi and Luca Cardelli. 1996. *A Theory of Objects*. Springer. https://doi.org/10.1007/978-1-4419-8598-9

[2] Andreas Abel and James Chapman. 2014. Normalization by Evaluation in the Delay Monad: A Case Study for Coinduction via Copatterns and Sized Types. In *Proceedings 5th Workshop on Mathematically Structured Functional Programming, MSFP@ETAPTS 2014 (Electronic Proceedings in Theoretical Computer Science, Vol. 153)*, Paul Levy and Neel Krishnaswami (Eds.). 51–67. https://doi.org/10.4204/EPTCS.153.4

[3] Peter Aczel. 1977. An Introduction to Inductive Definitions. In *Handbook of Mathematical Logic*, Jon Barwise (Ed.). Studies in Logic and the Foundations of Mathematics, Vol. 90. Elsevier, 739 – 782.

[4] Mads Sig Ager. 2004. From Natural Semantics to Abstract Machines. In *Logic-Based Program Synthesis and Transformation - 14th International Symposium, LOPSTR 2004 (Lecture Notes in Computer Science, Vol. 3573)*, Sandro Etalle (Ed.). Springer, Berlin, 245–261. https://doi.org/10.1007/11506676_16

[5] Nada Amin and Tiark Rompf. 2017. Type Soundness Proofs with Definitional Interpreters. In *The 44th Annual ACM Symposium on Principles of Programming Languages, POPL'17*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM Press, New York, 666–679. https://doi.org/10.1145/3009837

[6] Davide Ancona. 2014. How to Prove Type Soundness of Java-like Languages without Forgoing Big-Step Semantics. In *Proceedings of 16th Workshop on Formal Techniques for Java-like Programs, FTfJP'14*, David J. Pearce (Ed.). ACM Press, New York, 1:1–1:6. https://doi.org/10.1145/2635631.2635846

[7] Davide Ancona, Francesco Dagnino, Jurriaan Rot, and Elena Zucca. 2020. A big step from finite to infinite computations. *Science of Computer Programming* 197 (2020), 102492. https://doi.org/10.1016/j.scico.2020.102492

[8] Davide Ancona, Francesco Dagnino, and Elena Zucca. 2017. Generalizing Inference Systems by Coaxioms. In *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017 (Lecture Notes in Computer Science, Vol. 10201)*, Hongseok Yang (Ed.). Springer, Berlin, 29–55. https://doi.org/10.1007/978-3-662-54434-1_2

[9] Davide Ancona, Francesco Dagnino, and Elena Zucca. 2017. Reasoning on Divergent Computations with Coaxioms. *Proceedings of ACM on Programming Languages* 1, OOPSLA (2017), 81:1–81:26. https://doi.org/10.1145/3133905

[10] Davide Ancona, Francesco Dagnino, and Elena Zucca. 2018. Modeling Infinite Behaviour by Corules. In *32nd European Conference on Object-Oriented Programming, ECOOP 2018 (LIPIcs, Vol. 109)*, Todd D. Millstein (Ed.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Dagstuhl, 21:1–21:31. https://doi.org/10.4230/LIPIcs.ECOOP.2018.21

[11] Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de'Liguoro. 1995. Intersection and Union Types: Syntax and Semantics. *Information and Computation* 119, 2 (1995), 202–230. https://doi.org/10.1006/inco.1995.1086

[12] Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. 2013. *Lambda Calculus with Types*. Cambridge University Press, Cambridge.

[13] Lorenzo Bettini, Viviana Bono, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Betti Venneri. 2018. Java & Lambda: a Featherweight Story. *Logical Methods in Computer Science* 14, 3 (2018), 24 pages. https://doi.org/10.23638/LMCS-14(3:17)2018

[14] Martin Bodin, Philippa Gardner, Thomas P. Jensen, and Alan Schmitt. 2019. Skeletal semantics and their interpretations. *Proceedings of ACM on Programming Languages* 3, POPL (2019), 44:1–44:31. https://doi.org/10.1145/3290357

[15] Venanzio Capretta. 2005. General Recursion via Coinductive Types. *Logical Methods in Computer Science* 1, 2 (2005), 28 pages. https://doi.org/10.2168/LMCS-1(2:1)2005

[16] Venanzio Capretta, Tarmo Uustalu, and Varmo Vene. 2009. Corecursive Algebras: A Study of General Structured Corecursion. In *Formal Methods: Foundations and Applications, 12th Brazilian Symposium on Formal Methods, SBMF 2009 (Lecture Notes in Computer Science, Vol. 5902)*, Marcel Vinícius Medeiros Oliveira and Jim Woodcock (Eds.). Springer, 84–100. https://doi.org/10.1007/978-3-642-10452-7_7

[17] James Chapman, Tarmo Uustalu, and Niccolò Veltri. 2019. Quotienting the delay monad by weak bisimilarity. *Mathematical Structures in Computer Scienc* 29, 1 (2019), 67–92. https://doi.org/10.1017/S0960129517000184

[18] Arthur Charguéraud. 2013. Pretty-Big-Step Semantics. In *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013 (Lecture Notes in Computer Science, Vol. 7792)*, Matthias Felleisen and Philippa Gardner (Eds.). Springer, Berlin, 41–60. https://doi.org/10.1007/978-3-642-37036-6_3

[19] Luca Ciccone, Francesco Dagnino, and Elena Zucca. 2021. Flexible Coinduction in Agda. In *12th International Conference on Interactive Theorem Proving, ITP 2021 (LIPIcs, Vol. 193)*, Liron Cohen and Cezary Kaliszyk (Eds.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 13:1–13:19. https://doi.org/10.4230/LIPIcs.ITP.2021.13

[20] Bruno Courcelle. 1983. Fundamental Properties of Infinite Trees. *Theoretical Computer Science* 25 (1983), 95–169. https://doi.org/10.1016/0304-3975(83)90059-2

[21] Patrick Cousot and Radhia Cousot. 1992. Inductive Definitions, Semantics and Abstract Interpretations. In *The 19th Annual ACM Symposium on Principles of Programming Languages, POPL'92*, Ravi Sethi (Ed.). ACM Press, New York, 83–94. https://doi.org/10.1145/143165.143184

[22] Francesco Dagnino. 2019. Coaxioms: flexible coinductive definitions by inference systems. *Logical Methods in Computer Science* 15, 1 (2019). https://doi.org/10.23638/LMCS-15(1:26)2019

[23] Francesco Dagnino. 2021. *Flexible Coinduction.* Ph.D. Dissertation. DIBRIS, University of Genova. https://web.archive.org/web/20210214063202id_/https://iris.unige.it/retrieve/handle/11567/1035050/502494/phdunige_3767524.pdf

[24] Francesco Dagnino. 2021. Foundations of regular coinduction. *Logical Methods in Computer Science* 17 (2021). Issue 4. https://doi.org/10.46298/lmcs-17(4:2)2021

[25] Francesco Dagnino, Davide Ancona, and Elena Zucca. 2020. Flexible coinductive logic programming. *Theory and Practice of Logic Programming* 20, 6 (2020), 818–833. https://doi.org/10.1017/S147106842000023X Issue for ICLP 2020.

[26] Francesco Dagnino, Viviana Bono, Elena Zucca, and Mariangiola Dezani-Ciancaglini. 2020. Soundness Conditions for Big-Step Semantics. In *Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020 (Lecture Notes in Computer Science, Vol. 12075)*, Peter Müller (Ed.). Springer, 169–196. https://doi.org/10.1007/978-3-030-44914-8_7

[27] Ugo Dal Lago and Margherita Zorzi. 2012. Probabilistic operational semantics for the lambda calculus. *RAIRO Theoretical Informatics and Applications* 46, 3 (2012), 413–450. https://doi.org/10.1051/ita/2012012

[28] Nils Anders Danielsson. 2012. Operational Semantics using the Partiality Monad. In *Proceedings of the 17th ACM International Conference on Functional Programming, ICFP 2012*, Peter Thiemann and Robby Bruce Findler (Eds.). ACM Press, New York, 127–138. https://doi.org/10.1145/2364527.2364546

[29] Rocco De Nicola and Matthew Hennessy. 1984. Testing Equivalences for Processes. *Theoretical Computer Science* 34, 1 (1984), 83 – 133. https://doi.org/10.1016/0304-3975(84)90113-0

[30] Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro, and Adolfo Piperno. 1998. A Filter Model for Concurrent lambda-Calculus. *SIAM Journal of Computing* 27, 5 (1998), 1376–1419. https://doi.org/10.1137/S0097539794275860

[31] James Gosling, Bill Joy, Guy L. Steele, Gilad Bracha, and Alex Buckley. 2014. *The Java Language Specification, Java SE 8 Edition* (1st ed.). Addison-Wesley Professional, Boston.

[32] Grzegorz Grudzinski. 2000. A Minimal System of Disjunctive Properties for Strictness Analysis. In *ICALP Workshops*, José D. P. Rolim, Andrei Z. Broder, Andrea Corradini, Roberto Gorrieri, Reiko Heckel, Juraj Hromkovic, Ugo Vaccaro, and J. B. Wells (Eds.). Carleton Scientific, Waterloo, Ontario, Canada, 305–322.

[33] Atsushi Igarashi and Hideshi Nagira. 2007. Union Types for Object-Oriented Programming. *Journal of Object Technology* 6, 2 (2007), 47–68. https://doi.org/10.5381/jot.2007.6.2.a3

[34] Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. 2001. Featherweight Java: A Minimal Core Calculus for Java and GJ. *ACM Transactions on Programming Languages and Systems* 23, 3 (2001), 396–450. https://doi.org/10.1145/503502.503505

[35] Gilles Kahn. 1987. Natural Semantics. In *4th Annual Symposium on Theoretical Aspects of Computer Science, STACS'87 (Lecture Notes in Computer Science, Vol. 247)*, Franz-Josef Brandenburg, Guy Vidal-Naquet, and Martin Wirsing (Eds.). Springer, Berlin, 22–39. https://doi.org/10.1007/BFb0039592

[36] Xavier Leroy and Hervé Grall. 2009. Coinductive big-step operational semantics. *Information and Computation* 207, 2 (2009), 284–304. https://doi.org/10.1016/j.ic.2007.12.004

[37] Daniel Marino and Todd D. Millstein. 2009. A generic type-and-effect system. In *Proceedings of TLDI'09: ACM International Workshop on Types in Languages Design and Implementation*, Andrew Kennedy and Amal Ahmed (Eds.). ACM Press, 39–50. https://doi.org/10.1145/1481861.1481868

[38] Conor McBride. 2015. Turing-Completeness Totally Free. In *Mathematics of Program Construction - 12th International Conference, MPC 2015 (Lecture Notes in Computer Science, Vol. 9129)*, Ralf Hinze and Janis Voigtländer (Eds.). Springer, 257–275. https://doi.org/10.1007/978-3-319-19797-5_13

[39] Robin Milner. 1978. A Theory of Type Polymorphism in Programming. *J. Comput. System Sci.* 17, 3 (1978), 348–375. https://doi.org/10.1016/0022-0000(78)90014-4

[40] Keiko Nakata and Tarmo Uustalu. 2009. Trace-Based Coinductive Operational Semantics for While. In *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009 (Lecture Notes in Computer Science, Vol. 5674)*, Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel (Eds.). Springer, 375–390. https://doi.org/10.1007/978-3-642-03359-9_26

[41] Keiko Nakata and Tarmo Uustalu. 2010. A Hoare Logic for the Coinductive Trace-Based Big-Step Semantics of While. In *Programming Languages and Systems - 19th European Symposium on Programming, ESOP 2010 (Lecture Notes in Computer Science, Vol. 6012)*, Andrew D. Gordon (Ed.). Springer, 488–506. https://doi.org/10.1007/978-3-642-11957-6_26

[42] Keiko Nakata and Tarmo Uustalu. 2010. Resumptions, Weak Bisimilarity and Big-Step Semantics for While with Interactive I/O: An Exercise in Mixed Induction-Coinduction. In *Proceedings 7th Workshop on Structural Operational Semantics, SOS 2010 (Electronic Proceedings in Theoretical Computer Science, Vol. 32)*, Luca Aceto and Pawel Sobocinski (Eds.). 57–75. https://doi.org/10.4204/EPTCS.32.5

[43] Scott Owens, Magnus O. Myreen, Ramana Kumar, and Yong Kiam Tan. 2016. Functional Big-Step Semantics. In *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016 (Lecture Notes in Computer Science, Vol. 9632)*, Peter Thiemann (Ed.). Springer, Berlin, 589–615. https://doi.org/10.1007/978-3-662-49498-1_23

[44] Benjamin C. Pierce. 2002. *Types and programming languages.* MIT Press, Cambridge, Massachusetts.

[45] Maciej Piróg and Jeremy Gibbons. 2014. The Coinductive Resumption Monad. In *Proceedings of the 30th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2014 (Electronic Notes in Theoretical Computer Science, Vol. 308)*. Elsevier, 273–288. https://doi.org/10.1016/j.entcs.2014.10.015

[46] Gordon D. Plotkin. 1981. *A structural approach to operational semantics.* Technical Report. Aarhus University.

[47] Gordon D. Plotkin. 2004. A Structural Approach to Operational Semantics. *Journal of Logic and Algebraic Programming* 60-61 (2004), 17–139.

[48] Casper Bach Poulsen and Peter D. Mosses. 2017. Flag-based Big-step Semantics. *Journal of Logic and Algebraic Methods in Programming* 88 (2017), 174–190. https://doi.org/10.1016/j.jlamp.2016.05.001

[49] John C. Reynolds. 1972. Definitional interpreters for higher-order programming languages. In *ACM'72, Proceedings of the ACM annual conference*, Vol. 2. ACM Press, New York, 717–740.

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

[51] Davide Sangiorgi. 2011. *Introduction to Bisimulation and Coinduction.* Cambridge University Press, USA.

[52] Ross Tate. 2013. The sequential semantics of producer effect systems. In *The 40th Annual ACM Symposium on Principles of Programming Languages, POPL'13*, Roberto Giacobazzi and Radhia Cousot (Eds.). ACM Press, 15–26. https://doi.org/10.1145/2429069.2429074

[53] Andrew K. Wright and Matthias Felleisen. 1994. A Syntactic Approach to Type Soundness. *Information and Computation* 115, 1 (1994), 38–94. https://doi.org/10.1006/inco.1994.1093

[54] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. 2020. Interaction trees: representing recursive and impure programs in Coq. *Proceedings of ACM on Programming Languages* 4, The 47th Annual ACM Symposium on Principles of Programming Languages, POPL'20 (2020), 51:1–51:32. https://doi.org/10.1145/3371119