CIS 670: Coinductive Big-Step Operational Semantics

Chuen Hoa Koh

Abstract

In this project, we study the paper *Coinductive big-step operational semantics* by Xavier Leroy and Hervé Grall, published in *Information and Computation* in 2009 [3].

Our emphasis is on coinduction, and we use this to prove some results in the paper. While this is our own effort, the authors have already provided a complete Coq development of all results, so our work is not original.

Contents

1 Introduction

Traditionally, operational semantics of programming languages have been defined along two styles. One is the structural operational semantics (SOS) of Plotkin [5], commonly known as small-step operational semantics. A small-step semantics states how terms in the programming language reduce atomically, one step at a time, from term to term. The other form is the natural semantics of Kahn [2], and is commonly known as big-step semantics. A big-step semantics relates program terms directly to their evaluated end results, i.e. values, and is natural because terms can almost be seen as just syntax standing in for values.

In the language of a minimal lambda calculus, the following shows the difference between small-step and big-step semantics for lambda application.

Small-step reduction $\rightarrow$:

$$\frac{e_2 \rightarrow e_2'}{(\lambda x.e_1)\ e_2 \rightarrow (\lambda x.e_1)\ e_2'}$$

Big-step evaluation $\Rightarrow$:

$$\frac{e_1 \Rightarrow \lambda x.e_1' \qquad e_2 \Rightarrow v \qquad [v/x]e_1' \Rightarrow v'}{e_1\ e_2 \Rightarrow v'}$$

The advantages and disadvantages of big-step semantics both come from the loss of detail by relating terms with their values.

In a language where programs may not terminate, small-step semantics can express both terminating and non-terminating programs. Furthermore, when the language is equipped with a type system to rule out programs that "go wrong", small-step semantics allows one to prove a preservation theorem and a progress theorem á la Wright and Felleisen [6], which would together constitute a proof of type safety for the language by induction on the small-step semantics. Traditional big-step semantics, on the other hand, can only describe terminating programs and cannot distinguish between diverging programs and programs that "go wrong", in the sense of getting stuck with no evaluation rule applicable. Type safety cannot be directly expressed as preservation plus progress; the analogous statement of progress (every well-typed term evaluates to a value) is too strong to be true. The inability to express diverging evaluations means that results for small-step semantics typically become weaker or false for big-step semantics, *mutatis mutandis.*

On the other hand, big-step semantics is usually more convenient for applications like proving the correctness of program transformations or compilation schemes. For instance, a term in the source language may compile to multiple instructions in the target language. This makes it awkward to state and prove correctness using small-step semantics, since the two programs are not equivalent step-by-step. The source-target correspondence is hence easier to define in "bigger" steps, and is more easily proven by induction on big-step evaluation.

In their paper, the authors attempt to reconcile this by bringing attention to the use of coinductive big-step semantics, or more generally the use of coinduction and coinductive interpretations, for this purpose. It investigates two forms: the first, proposed by Cousot and Cousot [1], extends the usual big-step semantics with big-step coinductive rules defining divergence explicitly; the second, proposed by the authors, is obtained by reading the usual big-step evaluation rules in a coinductive way. Both forms allow one to now define the semantics of diverging programs in big-step fashion.

Similar extensions to small-step semantics can be made, so that one gets the following table.

Small-step	$a \to v$	$a \to^* v$	$a \xrightarrow{\infty}$	$a \xrightarrow{\text{co}*} v$
Big-step		$a \Rightarrow v$	$a \overset{\infty}{\Rightarrow}$	$a \overset{\text{co}}{\Rightarrow} v$

For small-step semantics, $\to$ is the single-step reduction, $\to^*$ is the reflexive transitive closure of $\to$, $\xrightarrow{\infty}$ defines divergence for reductions using coinductive rules, and the co-reduction relation $\xrightarrow{\text{co}*}$ is obtained by a coinductive interpretation of the small-step rules. Big-step evaluation, divergence, and co-evaluation are directly beneath their counterparts.

The paper investigates these semantics in the setting of a minimal lambda calculus, and makes the following contributions. Firstly, it studies the implication relationships between these semantics, in both the usual (trace-free) setting and a setting with trace semantics. Trace semantics describes the entire program execution, rather than just how program terms reduce/evaluate, and are thus useful for verifying transformations. These relationships show that one may define both a small-step and a big-step semantics for a language, and use whichever is more convenient for the task at hand; the (qualified) implications allow one to move between the different semantics.

Secondly, the paper illustrates how type soundness for a type system with recursive types can be expressed and proven using the different semantics. The recursive type system is given by interpreting typing rules coinductively (once again), and is necessary to preserve expressiveness of the language. For example, the call-by-value fixed-point combinator

$$Y = \lambda f.\, (\lambda x.\, f\,(x\,x))(\lambda x.\, f\,(\lambda y.\, (x\,x)\,y))$$

can have the type

$$((\tau \to \tau') \to \tau \to \tau') \to \tau \to \tau'$$

for all types τ and τ'. The general theorem we want is that every well-typed program either reduces/evaluates to a value or diverges (without getting stuck), and this is demonstrated for both the small-step and big-step forms. As Milner puts it, "well-typed programs don't go wrong".

Thirdly, the paper illustrates the use of coinductive interpretations in program compilation. It defines coinductive big-step semantics for a target language running on an abstract stack machine, gives a compilation scheme from the lambda calculus to this target language, and then proves that the compilation is correct. While the proof is not easy, it stands in contrast to the approach using small-step semantics, where one would usually be required to formulate a simulation relation between intermediate states of the abstract machine and the source language. This amounts to a sort of decompilation, and is very difficult to do in general settings. The coinduction proof works in greater generality, requiring just a recourse to an auxiliary coinductive relation which passes the guardedness check. This observation hence demonstrates the usefulness of big-step semantics in a setting like program transformations.

For this project, we study only the first part of the paper in detail. Specifically, we investigate (1) coinduction as a definitional and proof principle, and how it leads naturally to explicit formulations of divergence and infinite computation, in both small-step and big-step forms; (2) properties of the different (trace-free) semantics, and relationships between them. We give proofs for all theorems. Due to our lack of understanding of coinduction in Coq, our organization and proofs are different from those in the paper, but the deviation is expected to be only superficial.

2 Coinduction

As its name suggests, coinduction is dual to induction. As such, we can motivate and understand coinduction better by taking a more careful look at the better-known notion of induction. For our purposes, we shall view induction and coinduction somewhat narrowly in two ways: the first as constructive definitional schemes, and the second as proof techniques for objects defined through these definitional schemes. In what follows, we first consider induction from both of these angles, and then dualize the setting to obtain and explain coinduction.

Induction embodies the idea of starting at an initial point, and then repeatedly perform a number of constructions from this point. For example, in classic mathematical induction on numbers, one is contended with proving $P(0)$ and $P(n) \Rightarrow P(n+1)$, because this proves $P(n)$ for any given n, by applying the implication n times. Induction as a proof scheme formalizes this intuition, believed to be sound, and a proof by induction is the application of this scheme. The same applies when induction is used as a definitional scheme: start with some initial elements, and give (inference) rules for constructing "new" elements given "old" ones. The defined object is the set of all elements given by repeating the application of these rules on initial elements *ad infinitum*. By this intention, the set is the least (smallest) set that is closed with respect to the rules. As an example, a list of numbers may be defined as follows.

$$\frac{}{\mathit{nil}\ \text{list}}$$

$$\frac{l\ \text{list} \qquad n\ \text{nat}}{n :: l\ \text{list}}$$

A set of inference rules like the one above is called an inference system, and the set it defines is sometimes called an inductively defined set.

It is useful to think of an inductive definition as defining a subset in the set of terms constructed "freely" from the initial elements (e.g. *nil*) and the operators (e.g. ::). The power set of these terms with set union and intersection forms a lattice: every family of sets has a least upper bound given by union, and a greatest lower bound given by intersection. Furthermore, every inference rule R can be viewed as a set-theoretic monotone function F_R on sets of terms, i.e. for every $X \subseteq Y$ of terms, $F_R(X) \subseteq F_R(Y)$. This can be extended to the whole inference system, so that the collection of rules can be viewed as a monotone function. As an example, for the inference system above, its corresponding function is

$$F(S) = \{n :: t \mid t \in S \wedge n \text{ nat}\} \cup \{nil\}\,.$$

Accordingly, we can just take inference systems to be such functions.

The classic theorem of Knaster and Tarski states that the fixed points for these functions form a (sub-)lattice. In particular, fixed points exist and there is a least and greatest fixed point. Since an inductively defined set is the smallest set closed under application of the inference rules, it is a fixed point of the corresponding monotone functions, and hence the least fixed point. Specifically, if S is defined inductively by an inference system F,

$$S = \bigcup_{n=1}^{\infty} F^n(\emptyset)\,.$$

This corresponds to the usual intuition that we can look at the inference rules and collect elements starting from the empty set and axioms.

Since all elements of an inductively defined set are accessible via a finite sequence of applications of inference rules, there is an induction principle for proving properties for all elements in the set. From a presentation of its definition as an inference system, we can extract this induction scheme. For the above example, we get:

$$\frac{}{nil \in P}$$

$$\frac{l \in P \qquad n \text{ nat}}{nil :: l \in P}$$

Showing the two entailments allows us to infer that the set of lists is a subset of P.

Note that the rules here relate to the usual presentation of structural induction in the usual way: P here denotes the characteristic set of the predicate P.

We are now ready to dualize. Firstly, coinduction embodies the idea of already reaching some *final* point, and then repeatedly perform a number of checks. This sense of finality is illustrated most clearly by coinduction as a definitional principle. The following inference system defines the set of infinite lists:

$$\begin{array}{c} l \text{ stream} \qquad n \text{ nat} \\ \hline\hline n :: l \text{ stream} \end{array}$$

In comparison with the inference system for lists, this rule is almost the same as the second one there, but there is no counterpart to the first rule for lists. Over there, the first rule provided the comfort of having some initial list to start from, and the second rule generates all other lists by repetition. Here however, there is no obvious starting point for having some stream.

Rather, we begin at the set of all freely generated terms. This can be seen as an optimistic hypothesis, where we pretend that the set of streams is the entire set itself. Then we check for consistency by *eliminating* terms by *rule inversion*: if t is a stream, then it must be "derivable" from one of the inference rules; since there is only one rule, t must be of the form $n :: t'$ for some

t'. This immediately rules out any nat n from the set of all terms. The set of streams is the set obtained by repeating this elimination process *ad infinitum.* By virtue of this construction, the set is the greatest (largest) set that is *consistent* with the rules.

This explanation shows why the inference system defines the set of streams, i.e. infinite lists: every element that is not an infinite list eventually gets eliminated.[1] It also shows why coinduction has this sense of finality. Every element is assumed to be in the set unless demonstrated not; if there is a base case and one can build up the set upwards by repeating the application of rules, a coinductively defined set is like the final point in this process, beyond the first infinite accessible set given by induction. Indeed, every element in the set can never be eliminated and hence has an infinite derivation tree built up from rule inversions, which if read in the forward direction, is a truly infinite construction. While allowing such ill-founded derivations in the calculus may be psychologically uncomfortable, such a set is the greatest fixed point in the context of inference systems as monotone functions, and hence exists.

As a proof principle, induction applies because all terms in an inductively defined set are accessible. For coinduction on the other hand, it is elements *not* in the coinductively defined set that are "accessible": one proves exclusion by exhibiting a finite refutation, in the form of a finite sequence of rule inversions that result in "getting stuck". This means, classically, that a set S containing only terms that never get stuck under inversion must be a subset of the coinductively defined set. This is coinduction as a proof principle.

Given a coinductive definition, one may extract this systematically. For streams, we have:

$$\frac{l \in S \qquad n \text{ nat}}{\overline{n :: l \in S}}$$

In words, we want to show that "if $n :: l \in S$, then n nat and $l \in S$". If this is proven, the proof scheme lets us infer that S is a subset of the set of streams. Note how this scheme resembles the induction proof scheme for lists, but we read the inference backwards and the conclusion that we get is the opposite set inclusion. Also, while the induction principle requires showing the property P of interest to be closed under all rules, the coinduction principle only requires showing that S is consistent with *some* subset of the rules, rather than all.

We have thus introduced coinduction as a dual concept to induction, as both definitional and proof schemes. However, while coinduction seems useful as a tool to define infinite objects, it is perhaps less clear how coinduction as a proof principle is useful. To illustrate this, we consider the bisimulation relation $\sim$ used to express process "equality" [4]; because processes can run forever, the notion of process equality is challenging to define, and bisimulation serves as a reasonable definition in this context. The relation can be cast naturally as the greatest fixed point of the following inference rule:

$$\frac{P_1 \to a.P_1' \qquad P_2 \to a.P_2' \qquad P_1' \sim P_2'}{\overline{P_1 \sim P_2}}$$

Here, the notation $P \to a.P'$ means that the process P emits a as an observation, and then becomes the process P'. The rule hence states that P_1 and P_2 are observationally equivalent "forever". With this definition of process equivalence, we may have the problem of proving that two processes P_1 and P_2 are observationally equivalent. The coinduction proof principle states that if we can show

[1]Of course, one may contend that terms generated "freely" from the syntax involved cannot be infinite, but this is where our informality rears its ugly head and the meta-theory should be made precise. Nevertheless, the explanation should make sense as a first approximation.

$$\frac{P_1 \to a.P_1' \qquad P_2 \to a.P_2' \qquad (P_1', P_2') \in S \times S}{(P_1, P_2) \in S \times S}$$

we would have shown that $S \times S \subseteq \sim$, i.e. all processes in S are observationally equivalent. In other words, it suffices to find some S that contains the two processes P_1 and P_2 of interest. This is a subtle point: if we try to prove $P_1 \sim P_2$ directly by inverting the rule (equivalently, to observe them), we would have to go on forever. Coinduction lets us stop, because an S that works must contain P_1' and P_2', which we would have to check in the first inversion; once checked, inversion can happen *ad infinitum* in principle.

Before we move on, we should note that coinductive proofs in Coq are usually done in a more direct manner by applying the `cofix` tactic, which builds the proof by guarded co-recursion and not through the associated coinduction principle. To prove a coinductive goal H, we can assume H itself, but ensure that the proof of H uses a constructor of H (i.e. guarded). This just corresponds to using an inference rule of H. If so, we can keep inverting and get an infinite proof. The paper uses coinduction in this proof-theoretic manner.

Even though understood as such, the `cofix` tactic applies to rather general forms, and we are not sure of its precise formalism. Hence we have avoided its use in this work, and all proofs by coinduction follow the form described in this section.

3 Semantics

The language under consideration is given by the following grammar.

t	::=		term
	$\vert$	x	variable
	$\vert$	c	constant
	$\vert$	$\lambda x.t$	lambda abstraction
	$\vert$	$t\, t'$	function application

While we have not stated its semantics in any form, they shall just be formalizing the usual β rule. Then this language, while minimal, already contains three behaviors of interest; a term/program in this language either:

- Computes to a value, which is defined to be a constant or a lambda abstraction; or
- Computes to a term that gets stucks and hangs; or
- Never terminate.

The problem with big-step semantics is the inability to distinguish between the second and third case: $\neg(a \Rightarrow v)$ admits both of them.

One workaround is to explicitly define the second in the form of an error/goes-wrong relation, after which we can express all three cases. However, the error relation is unnatural and awkward to define explicitly because it is not how computation proceeds (in fact, the opposite), and because it is combinatorially large in the number of syntax combinators. The latter means that the definition is prone to human error, especially in the form of omitted cases. Theorems of the form "if P holds,

e never goes wrong" may well give false confidence; in fact, if "going wrong" is the empty relation with all cases accidentally omitted, the theorem holds as long as P holds!

A better way therefore, is to explicitly define divergence, i.e. the third case, instead. In this section, we motivate and introduce various semantics from this as starting point, prove results which clarify what they really define, and prove relationships that hold among them.

For proofs, we follow the convention of marking them with "Classical" whenever we have to use the law of excluded middle. While the authors did so because the development is in Coq, we are just going along for purist reasons.

3.1 Small-Step Reduction and its Kleene Closure

We start off by considering small-step reduction $\rightarrow$, which is defined in the straightforward way. To gain convenience by reasoning in larger steps, one may naturally extend this by taking the reflexive transitive closure $\rightarrow^*$ of $\rightarrow$. The rules on the left define $\rightarrow$, while the rules on the right define $\rightarrow^*$.

$$\frac{v \text{ value}}{(\lambda x.\, b)\, v \rightarrow [v/x]b}\ \rightarrow\text{-}\beta \qquad\qquad \frac{}{a \rightarrow^* a}\ \rightarrow^*\text{-refl}$$

$$\frac{a_1 \rightarrow a_1'}{a_1\, a_2 \rightarrow a_1'\, a_2}\ \rightarrow\text{-app-l} \qquad\qquad \frac{a \rightarrow a' \qquad a' \rightarrow^* b}{a \rightarrow^* b}\ \rightarrow^*\text{-trans}$$

$$\frac{a_1 \text{ value} \qquad a_2 \rightarrow a_2'}{a_1\, a_2 \rightarrow a_1\, a_2'}\ \rightarrow\text{-app-r}$$

We first establish some basic properties of this relation.

Proposition 3.1. *If v is a value, $v \not\rightarrow$, i.e. there is no a such that $v \rightarrow a$.*

Proof. By simply noting that no rule applies. □

Proposition 3.2. *For all a, a_1, a_2, if $a \rightarrow a_1$ and $a \rightarrow a_2$, then $a_1 = a_2$.*

Proof. By induction on a. The only case that applies is where $a = t_1\, t_2$.

We then take cases on t_1 and t_2 being values or not.

Case: When t_1 is not a value, only $\rightarrow$-app-l applies. By inversion on $a \rightarrow a_1$ and $a \rightarrow a_2$, we have $t_1 \rightarrow b_1$ and $t_1 \rightarrow b_2$ for some b_1 and b_2. By the induction hypothesis, $b_1 = b_2$, so $a_1 = b_1\, t_2 = b_2\, t_2 = a_2$.

Case: When t_1 is a value and t_2 is not a value, only $\rightarrow$-app-r applies. By inversion again, we have $t_2 \rightarrow b_1$ and and $t_2 \rightarrow b_1$ for some b_1 and b_2. By the induction hypothesis, $b_1 = b_2$, so $a_1 = t_1\, b_1 = t_1\, b_2 = a_2$.

Case: When t_1 and t_2 are both values, only $\rightarrow$-β applies. In that case, $t_1 = (\lambda x.\, b)$ for some b and $t_2 = v$ for some value v. Determinism of substitution then implies the claim.

This covers all cases, so the claim follows. □

Since every step of the small-step reduction is deterministic, $\rightarrow^*$ is also deterministic; every computation goes along a single path.

Proposition 3.3. *For all a, a_1, a_2, if $a \rightarrow^* a_1$ and $a \rightarrow^* a_2$, either $a_1 \rightarrow^* a_2$ or $a_2 \rightarrow^* a_1$.*

Proof. By induction on $a \to^* a_1$.

Case: For the reflexive case, we have $a = a_1$. Then $a_1 \to^* a_2$ trivially.

Case: For the transitive case, there is some a' such that $a \to a'$ and $a' \to^* a_1$. If the reflexive case holds for $a \to^* v_2$, we have $a_2 \to^* a_1$. If it is the transitive case that holds on the other hand, there exists a'' such that $a \to a''$ and $a'' \to^* a_2$. By determinism of $\to$, $a' = a''$. By the induction hypothesis, either $a_1 \to^* a_2$ or $a_2 \to^* a_1$. □

Corollary 3.4. *For all a and values v_1, v_2, if $a \to^* v_1$ and $a \to^* v_2$, $v_1 = v_2$.*

Proof. Immediate from the previous proposition. □

We assume the following fact for reflexive transitive closures, which is the stronger form of the $\to^*$-trans rule.

Proposition 3.5. *If $a \to^* a'$ and $a' \to^* b$, then $a \to^* b$.*

We now show that $\to^*$ can replace $\to$ in all its rules.

Proposition 3.6. *If $a_1 \to^* a_1'$, then $a_1\, a_2 \to^* a_1'\, a_2$.*

Proof. By induction on $\to^*$. If $a_1 = a_1'$ following $\to^*$-refl, just use $\to^*$-refl to prove the claim.

If $a_1 \to a_1''$ and $a_1'' \to^* a_1'$, we have by the induction hypothesis that $a_1''\, a_2 \to^* a_1'\, a_2$. By $\to$-app-l, $a_1\, a_2 \to a_1''\, a_2$. Then by $\to^*$-trans, $a_1\, a_2 \to^* a_1'\, a_2$. □

Proposition 3.7. *If a_1 is a value and $a_2 \to^* a_2'$, $a_1\, a_2 \to^* a_1\, a_2'$.*

Proof. By induction on $a_2 \to^* a_2'$. Again, the base case is trivial. For $a_2 \to a_2''$ and $a_2'' \to^* a_2'$, again use the induction hypothesis to deduce $a_1\, a_2'' \to^* a_1\, a_2'$. Since a_1 is a value, by $\to$-app-r, $a_1\, a_2 \to a_1\, a_2''$. By $\to^*$-trans, $a_1\, a_2 \to^* a_1\, a_2'$, as desired. □

Proposition 3.8. *If $a_1 \to^* \lambda x.\, b$, $a_2 \to^* v$ for some v, then $a_1\, a_2 \to^* (\lambda x.\ b) v$.*

Proof. First by induction on $a_1 \to^* \lambda x.\, b$.

Case: $a_1 = \lambda x.\, b$. Apply Proposition 3.7.

Case: $a_1 \to a_1'$ and $a_1' \to^* \lambda x.\, b$. By the induction hypothesis, $a_1'\, a_2 \to^* (\lambda x.\, b)\, v$. We have by $\to$-app-l that $a_1\, a_2 \to a_1'\, a_2$. Then apply $\to^*$-trans to conclude the claim. □

Corollary 3.9. *If $a_1 \to^* \lambda x.\, b$, $a_2 \to^* v$ for some v, and $[v/x]b \to^* v'$, then $a_1\, a_2 \to^* v'$.*

Proof. Use Proposition 3.8 to show that $a_1\, a_2 \to^* (\lambda x.\, b)\, v$. By $\to$-app-β, $(\lambda x.\, b)\, v \to [v/x]b$, and hence $(\lambda x.\, b)\, v \to^* [v/x]b$. Since $[v/x]b \to^* v'$, we can chain up the three using Proposition 3.5 to prove the claim. □

In other words, the following rules all hold (but only in the forward direction).

$$\frac{v \text{ value}}{(\lambda x.\, b)\, v \to^* [v/x]b} \to^*\text{-}\beta \qquad \frac{a_1 \to^* a_1'}{a_1\, a_2 \to^* a_1'\, a_2} \to^*\text{-app-l}$$

$$\frac{a_1 \text{ value} \qquad a_2 \to^* a_2'}{a_1\, a_2 \to^* a_1\, a_2'} \to^*\text{-app-r}$$

3.2 Co-reduction and Small-Step Divergence

The above excursion should show that small-step $\to$ can express all computation outcomes: $a \to^* v$ states that a computes to a value, and $a \not\to a'$ for all a' expresses getting stuck. The third case of divergence is handled by mutual exclusion from the first two. This is already a clear advantage over big-step semantics.

Even so, we may try to define small-step divergence directly. The first observation is that $\to^*$ only captures finitely many reductions and not divergence, because every proof of $a \to^* a'$ must be finite. Once we allow coinduction in the meta-theory however, we may try to circumvent this by reading exactly the same rules coinductively. This defines the co-reduction relation $a \xrightarrow{\text{co}*} a'$ on pairs of terms.

$$\frac{}{a \xrightarrow{\text{co}*} a} \xrightarrow{\text{co}*}\text{-refl} \qquad \frac{a \to a' \qquad a' \xrightarrow{\text{co}*} b}{a \xrightarrow{\text{co}*} b} \xrightarrow{\text{co}*}\text{-trans}$$

A careful look at these rules show two facts. Firstly, the reflexive rule can be applied infinitely often to any a, and hence twarts the attempt to capture divergence. Indeed, the presence of the reflexive rule means that co-reduction includes both termininating *and* diverging terms, and saying $a \xrightarrow{\text{co}*} a'$ admits both the possibility of a diverging and of a reducing to a' in a finite number of steps. Secondly, even if we discard the reflexive rule to force an infinite application of the transitive rule, $\xrightarrow{\text{co}*}$ is somewhat awkward as a two-place relation, since the a' in $a \xrightarrow{\text{co}*} a'$ can play no role if a diverges.

In view of this, we define divergence explicitly using the following rule instead. It defines the small-step divergence relation $\xrightarrow{\infty}$.

$$\frac{a \to a' \qquad a' \xrightarrow{\infty}}{a \xrightarrow{\infty}} \xrightarrow{\infty}\text{-trans}$$

The following propositions show that that $\xrightarrow{\infty}$ indeed describes divergence.

Proposition 3.10. *Suppose that whenever $a \to^* a'$, there exists b such that $a' \to b$. Then $a \xrightarrow{\infty}$.*

Proof. By coinduction on $a \xrightarrow{\infty}$. Let $S(a)$ be the statement

"For all a', $a \to^* a'$ implies there exists b such that $a' \to b$."

Let a be such that $S(a)$ holds. Since $a \to^* a$ by $\to^*$-refl, there exists a' such that $a \to a'$. Let b such that $a' \to^* b$. Then $a \to^* b$ by $\to^*$-trans. By membership in S, there is b' such that $b \to b'$. Hence $a' \in S$ too. This shows that S is consistent with $\xrightarrow{\infty}$, and the claim follows by the principle of coinduction. □

Proposition 3.11. *For all a and a', if $a \xrightarrow{\infty}$ and $a \to^* a'$, then $a' \xrightarrow{\infty}$.*

Proof. By induction on $a \to^* a'$. The claim is trivial for $a = a'$. For the inductive step, suppose $a \to a''$ and $a'' \to^* a'$. By inversion on $a \xrightarrow{\infty}$ and Proposition 3.2, $a'' \xrightarrow{\infty}$. By the induction hypothesis, $a' \xrightarrow{\infty}$. $\square$

Corollary 3.12. *For any a and a', if $a \xrightarrow{\infty}$ and $a \to^* a'$, then $a' \to$, i.e. there exists b such that $a' \to b$.*

Proof. By Proposition 3.11 and inversion on $a' \xrightarrow{\infty}$. $\square$

Proposition 3.13. *For all a, $a \xrightarrow{\infty}$ if and only if for all a', whenever $a \to^* a'$, $a' \to$.*

Proof. By Proposition 3.10 and Corollary 3.12. $\square$

If we allow classical reasoning, then every term either diverges or gets stuck, as stated at the beginning of this section.

Proposition 3.14. *For all a, either $a \xrightarrow{\infty}$ or there exists b such that $a \to^* b$ and $b \not\to$.*

Proof. **(Classical)** By the law of excluded middle, on whether b exists or not. If not, apply Proposition 3.13. $\square$

The two relations $\xrightarrow{\text{co}*}$ and $\xrightarrow{\infty}$ are clearly very similar, and the following results show exactly how they are.

Proposition 3.15. *If $a \to a'$ and $a \xrightarrow{co*} b$, then $a' \xrightarrow{co*} b$.*

Proof. Case analysis on $a \xrightarrow{\text{co}*} b$. $\square$

Proposition 3.16. *For all a and a', if $a \xrightarrow{co*} a'$ and $a \not\to^* a'$, then $a \xrightarrow{\infty}$.*

Proof. By coinduction on $a \xrightarrow{\infty}$.

Let $S(a)$ be the statement

“There exists a' such that $a \xrightarrow{\text{co}*} a'$ and $a \not\to^* a'$.”

Let a be such that $S(a)$ holds, i.e. there exists a' such that $a \xrightarrow{\text{co}*} a'$ and $a \not\to^* a'$. By inversion on $a \xrightarrow{\text{co}*} a'$, either $a = a'$ or there exists b such that $a \to b$ and $b \xrightarrow{\text{co}*} a'$. The first case is trivial. For the second, we just note that if $b \to^* a'$, then $a \to^* a'$, a contradiction. So $b \not\to^* a'$. This says that $S(b)$ holds, and the claim follows by coinduction. $\square$

The following shows that co-reduction means either finite reduction or infinite reduction.

Proposition 3.17. *For all a and a', if $a \xrightarrow{co*} a'$, either $a \to^* a'$ or $a \xrightarrow{\infty}$.*

Proof. **(Classical)** Follows from the above by the law of excluded middle. $\square$

For the reverse, we first observe that small-step divergence implies co-reduction to any value.

Proposition 3.18. *For all a and a', $a \xrightarrow{\infty}$ implies $a \xrightarrow{co*} a'$ for all a'.*

Proof. Fix a', and then prove by coinduction on $a \xrightarrow{\text{co}*} a'$. It suffices to show that $a \xrightarrow{\infty}$ is consistent with $\xrightarrow{\text{co}*}$-trans, i.e. if $a \xrightarrow{\infty}$ and $a \to a'$, then $a' \xrightarrow{\infty}$. But this is just the inversion of $\xrightarrow{\infty}$-trans. $\square$

Proposition 3.19. *For all a and a', $a \to^* a'$ implies $a \xrightarrow{co*} a'$.*

Proof. By induction on $\to^*$. When $a = a'$, just use $\xrightarrow{\text{co}*}$-refl. When $a \to a''$ and $a'' \to^* a'$, $a'' \xrightarrow{\text{co}*} a'$ by the induction hypothesis, and the claim follows by $\xrightarrow{\text{co}*}$-trans. ☐

Summarizing, the relationship between $\xrightarrow{\text{co}*}$ and $\xrightarrow{\infty}$ is as follows.

Proposition 3.20. *For all a and a', $a \xrightarrow{\text{co}*} a'$ if and only if $a \to^* a'$ or $a \xrightarrow{\infty}$.*

Proof. By Proposition 3.17, Proposition 3.18, and Proposition 3.19. ☐

3.3 Big-Step Evaluation and Big-Step Divergence

We now switch attention to big-step semantics, which is given by the following rules.

$$\frac{}{c \Rightarrow c}\ \Rightarrow\text{-const} \qquad\qquad \frac{}{\lambda x.\, a \Rightarrow \lambda x.\ a}\ \Rightarrow\text{-fun}$$

$$\frac{a_1 \Rightarrow \lambda x.\, b \qquad a_2 \Rightarrow v \qquad [v/x]b \Rightarrow v'}{a_1\ a_2 \Rightarrow v'}\ \Rightarrow\text{-app}$$

Proposition 3.21. *If $a \to a'$ and $a' \Rightarrow v$ for some value v, then $a \Rightarrow v$.*

Proof. By induction on $a \to a'$.

Case: $a = (\lambda x.\, b)\, v'$ and $a' = [v'/x]b$ for some b' and value v'. Since $\lambda x.\, b \Rightarrow \lambda x.\, b$, $v' \Rightarrow v'$, and $[v'/x] \Rightarrow v$, we can apply $\Rightarrow$-app to conclude $a \Rightarrow v$.

Case: $a = a_1\, a_2$, $a_1 \to a_1'$, and $a' = a_1'\, a_2$. Since $a_1'\, a_2 \Rightarrow v$, only $\Rightarrow$-app is applicable, so we have by rule inversion that $a_1' \Rightarrow \lambda x.\, b$, $a_2 \Rightarrow v'$ and $[v'/x]b \Rightarrow v$, for some b and value v'.

By the induction hypothesis, since $a_1 \to a_1'$ and $a_1' \Rightarrow \lambda x.\, b$, $a_1 \Rightarrow \lambda x.\ b$. Then apply $\Rightarrow$-app to conclude that $a_1\, a_2 \Rightarrow v$, as desired.

Case: $a = a_1\, a_2$, a_1 is a value, $a_2 \to a_2'$, and $a' = a_1\, a_2'$. Since $a_1\, a_2' \Rightarrow v$, we again have by rule inversion that $a_1 \Rightarrow \lambda x.\, b$, $a_2' \Rightarrow v'$, and $[v'/x]b \Rightarrow v$ for some b and value v'. Since $a_2 \to a_2'$ and $a_2' \Rightarrow v'$, the induction hypothesis implies $a_2 \Rightarrow v'$. Then apply $\Rightarrow$-app to conclude the claim. ☐

The following result shows why big-step semantics is "big-step".

Proposition 3.22. *For all a and value v, $a \Rightarrow v$ if and only if $a \to^* v$.*

Proof. For the forward direction, we do induction on $\Rightarrow$. When a is a constant or a lambda abstraction, we use $\to^*$-refl. When $a = a_1\, a_2$, such that $a_1 \Rightarrow \lambda x.\, b$ and $a_2 \Rightarrow v$ for some b and value v, and $[v/x]b \Rightarrow v'$, first apply the induction hypothesis to get $a_1 \to^* \lambda x.\ b$, $a_2 \to^* v$, and $[v/x]b \to^* v'$. By Corollary 3.9, $a_1\, a_2 \to^* v'$, so the claim follows.

For the reverse direction, we do induction on $a \to^* v$. The base case is trivial, since $\Rightarrow$-const and $\Rightarrow$-fun implies that $v \Rightarrow v$ for all values v. If $a \to a'$ and $a' \to^* v$, the induction hypothesis implies that $a' \Rightarrow v$, and Proposition 3.21 yields the claim. ☐

As a corollary, we see that big-step semantics is also deterministic.

Corollary 3.23. *If $a \Rightarrow v_1$ and $a \Rightarrow v_2$, $v_1 = v_2$.*

Proof. By Corollary 3.4 and Proposition 3.22. ☐

Big-step divergence is defined by the following inference system.

$$\frac{a_1 \overset{\infty}{\Rightarrow}}{a_1\ a_2 \overset{\infty}{\Rightarrow}}\ \overset{\infty}{\Rightarrow}\text{-app-l} \qquad\qquad \frac{a_1 \Rightarrow v \qquad a_2 \overset{\infty}{\Rightarrow}}{a_1\ a_2 \overset{\infty}{\Rightarrow}}\ \overset{\infty}{\Rightarrow}\text{-app-r}$$

$$\frac{a_1 \Rightarrow \lambda x.\, b \qquad a_2 \Rightarrow v \qquad [v/x]b \overset{\infty}{\Rightarrow}}{a_1\ a_2 \overset{\infty}{\Rightarrow}}\ \overset{\infty}{\Rightarrow}\text{-app-f}$$

A first observation is the following.

Proposition 3.24. *For all values v, $\neg(v \overset{\infty}{\Rightarrow})$.*

Proof. By inspection of the rules. □

Big-step divergence implies small-step divergence, as the following shows.

Proposition 3.25. *For all a, $a \overset{\infty}{\Rightarrow}$ implies $a \xrightarrow{\infty}$.*

Proof. By coinduction, it suffices to show that $a \overset{\infty}{\Rightarrow}$ is consistent with $a \xrightarrow{\infty}$. This in turn can be proven by induction on a. The cases for a being a constant, a variable, or a lambda abstraction do not apply since $\overset{\infty}{\Rightarrow}$ applies only to function applications, so the claim holds trivially in these cases.

So $a = a_1\, a_2$ for some a_1 and a_2. Since $a_1\, a_2 \overset{\infty}{\Rightarrow}$, we have three cases by inversion.

Case: $a_1 \overset{\infty}{\Rightarrow}$. Then by the induction hypothesis, there exists a_1' such that $a_1 \to a_1'$ and $a_1' \overset{\infty}{\Rightarrow}$. By $\overset{\infty}{\Rightarrow}$-app-l, $a_1'\, a_2 \overset{\infty}{\Rightarrow}$. By $\to$-app-l, $a_1\, a_2 \to a_1'\, a_2$.

Case: $a_1 \Rightarrow v$ for some value v and $a_2 \overset{\infty}{\Rightarrow}$. By Proposition 3.22, $a_1 \to^* v$, so either $a_1 = v$ or there exists a_1' such that $a_1 \to a_1'$ and $a_1' \to^* v$.

For the first case, the induction hypothesis implies that there exists a_2' such that $a_2 \to a_2'$ and $a_2' \overset{\infty}{\Rightarrow}$. Apply $\overset{\infty}{\Rightarrow}$-app-r to conclude $a_1\, a_2' \overset{\infty}{\Rightarrow}$. By $\to$-app-r, $a_1\, a_2 \to a_1\, a_2'$.

For the second case, Proposition 3.22 implies $a_1' \Rightarrow v$. Then apply $\overset{\infty}{\Rightarrow}$-app-r to conclude $a_1'\, a_2 \overset{\infty}{\Rightarrow}$. By $\to$-app-l, $a_1\, a_2 \to a_1'\, a_2$.

Case: $a_1 \Rightarrow \lambda x.\, b$, $a_2 \Rightarrow v$, and $[v/x]b \overset{\infty}{\Rightarrow}$, for some b and value v. By $\to$-β, $a_1\, a_2 \to [v/x]b$.

We have thus shown that $\overset{\infty}{\Rightarrow}$ is consistent with $\xrightarrow{\text{co}*}$ in all cases, so the claim holds by coinduction. □

The converse implication also turns out to hold.

Proposition 3.26. *For all a, $a \xrightarrow{\infty}$ implies $a \overset{\infty}{\Rightarrow}$.*

Proof. (**Classical**) By coinduction on $a \overset{\infty}{\Rightarrow}$, it suffices to show that $a \xrightarrow{\infty}$ is consistent with $a \overset{\infty}{\Rightarrow}$.

We prove this by taking cases on a. The base cases where a is a constant, variable, or lambda abstraction are all trivial since $a \xrightarrow{\infty}$ implies that $a \to a'$ for some a', but $a \not\to$ for these cases.

So suppose $a = a_1\, a_2$. By Proposition 3.14 (classical), either $a_1 \xrightarrow{\infty}$ or there exists b_1 such that $a_1 \to^* b_1$ and $b_1 \not\to$. The first case immediately gives consistency with $\overset{\infty}{\Rightarrow}$-app-l.

For the second case, we have by Proposition 3.7 that $a_1\, a_2 \to^* b_1\, a_2$. By Proposition 3.11, $b_1\, a_2 \xrightarrow{\infty}$. By inversion on this, $b_1\, a_2 \to$. We hence have two cases:

Case: b_1 is a value and $a_2 \to a_2'$ for some a_2'. Then again take cases using Proposition 3.14:

- $a_2 \xrightarrow{\infty}$: By Proposition 3.22, $a_1 \Rightarrow b_1$, a value. Hence we have consistency with $\overset{\infty}{\Rightarrow}$-app-r.

- $a_2 \to^* b_2$ and $b_2 \not\to$: By Proposition 3.7, $b_1\ a_2 \to^* b_1\ b_2$. By Corollary 3.12, since $b_1\, a_2 \xrightarrow{\infty}$, we have $b_1\ b_2 \to$, and by inversion the only possibility left is that $b_1 = \lambda x.\, b$ for some b and $b_2 = v$ for some value v. By Proposition 3.22, $a_1 \Rightarrow \lambda x.\, b$ and $a_2 \Rightarrow v$. This is the same as the second case below.

Case: $b_1 = \lambda x.\, b$ for some b and and $a_2 = v$ for some value v. Then by Proposition 3.22, $a_1 \Rightarrow \lambda x.\, b$. Also, $a_2 \Rightarrow v$ since it is a value. By $\to$-β, $b_1\, a_2 \to [v/x]b$. Since $b_1\, a_2 \xrightarrow{\infty}$, another application of Proposition 3.11 shows that $[v/x]b \xrightarrow{\infty}$. So we have consistency with $\overset{\infty}{\Rightarrow}$-app-f.

This covers all the cases and completes the proof. □

In summary, small-step and big-step divergence coincide. Since Proposition 3.13 already convinces us that small-step divergence describes divergent terms, big-step divergence does too.

Proposition 3.27. *For all a, $a \overset{\infty}{\Rightarrow}$ if and only if $a \xrightarrow{\infty}$.*

Proof. By Proposition 3.25 and Proposition 3.26. □

Through small-step semantics, we also observe that terms cannot both evaluate to a value and diverge.

Proposition 3.28. *If $a \Rightarrow v$ for some v, $a \overset{\infty}{\not\Rightarrow}$. if $a \overset{\infty}{\Rightarrow}$, then there is no v such that $a \Rightarrow v$.*

Proof. For the first, $a \to^* v$ by Proposition 3.22. By Corollary 3.12, $a \overset{\infty}{\not\to}$. By Proposition 3.25, $a \overset{\infty}{\not\Rightarrow}$.

For the second, Proposition 3.25 implies $a \xrightarrow{\infty}$. If v exists, Proposition 3.22 tells us $a \to^* v$. Since $v \not\to$, we get a contradiction from Corollary 3.12. □

It is also possible to prove this directly using big-step semantics without recourse to small-step equivalence results. (This is the approach taken in the paper, which works through big-step semantics before small-step; see Lemma 6.)

We can now illustrate an application for divergence semantics. Let

$$\begin{aligned} \omega &= \delta\, \delta \\ \delta &= \lambda x.\, x\, x \end{aligned}$$

It is easy to informally check that ω reduces infinitely. Given our definitions, we can now express this formally.

Proposition 3.29. *$\omega \overset{\infty}{\Rightarrow}$ and $\omega \xrightarrow{\infty}$.*

Proof. Let $S = \{\omega\}$. For the first, apply coinduction and note consistency with $\overset{\infty}{\Rightarrow}$-app-f. For the second, apply coinduction and compute $\to$-β. □

In light of having investigated $\xrightarrow{\text{co}*}$, one may also be curious about interpreting big-step semantics coinductively. This gives us $\overset{\text{co}}{\Rightarrow}$, the co-evaluation relation.

$$\frac{}{c \overset{\text{co}}{\Rightarrow} c}\ \overset{\text{co}}{\Rightarrow}\text{-const} \qquad\qquad \frac{}{\lambda x.\, a \overset{\text{co}}{\Rightarrow} \lambda x.\ a}\ \overset{\text{co}}{\Rightarrow}\text{-fun}$$

$$\frac{a_1 \overset{\text{co}}{\Rightarrow} \lambda x.\, b \qquad a_2 \overset{\text{co}}{\Rightarrow} v \qquad [v/x]b \overset{\text{co}}{\Rightarrow} v'}{a_1\ a_2 \overset{\text{co}}{\Rightarrow} v'}\ \overset{\text{co}}{\Rightarrow}\text{-app}$$

Like co-reduction, co-evaluation captures both finite and diverging computations. The following is a first hint.

Proposition 3.30. *For all a and value v, if $a \overset{\text{co}}{\Rightarrow} v$, either $a = v$ or there exists a' such that $a \to a'$ and $a' \overset{\text{co}}{\Rightarrow} v$.*

Proof. By induction on a. When a is a constant or lambda abstraction, we note that only $\overset{\text{co}}{\Rightarrow}$-const and $\overset{\text{co}}{\Rightarrow}$-fun can apply respectively, and both of these rules are reflexive. If a is a variable, no rule applies so the claim holds vacuously.

So let $a = a_1\, a_2$ for some a_1 and a_2. Only $\overset{\text{co}}{\Rightarrow}$-app can apply, so by inversion, $a_1 \overset{\text{co}}{\Rightarrow} \lambda x.\, b$, $a_2 \overset{\text{co}}{\Rightarrow} v'$, and $[v'/x]b \overset{\text{co}}{\Rightarrow} v$ for some b and value v'. By the induction hypothesis, either $a_1 = \lambda x.\, b$ or $a_1 \to a_1'$ and $a_1' \overset{\text{co}}{\Rightarrow} \lambda x.\, b$ for some a_1'. If it is the second, we have $a_1\, a_2 \to a_1'\, a_2$ by $\to$-app-l, and we can use $\overset{\text{co}}{\Rightarrow}$-app to conclude that $a_1'\, a_2 \overset{\text{co}}{\Rightarrow} v$.

If it is the first, we apply the induction hypothesis for a_2 to conclude that either $a_2 = v'$ or $a_2 \to a_2'$ and $a_2' \overset{\text{co}}{\Rightarrow} v'$ for some a_2'. Again, the second case is easy: use $\to$-app-r to conclude that $a_1\, a_2 \to a_1\, a_2'$, and $\overset{\text{co}}{\Rightarrow}$-app to conclude that $a_1\, a_2' \overset{\text{co}}{\Rightarrow} v$.

So we are left with $a_1 = \lambda x.\, b$ and $a_2 = v'$. In this case, we use $\to$-β to show $a_1\, a_2 \to [v'/x]b$, and since $[v'/x]b \overset{\text{co}}{\Rightarrow} v$, the claim follows.

This covers all the cases and completes the proof. □

Indeed, having defined $\overset{\infty}{\Rightarrow}$, we can express the latter possibility in terms of that. The following is akin to Proposition 3.17 for co-reduction.

Proposition 3.31. *For any a and value v, whenever $a \overset{\text{co}}{\Rightarrow} v$, either $a \Rightarrow v$ or $a \overset{\infty}{\Rightarrow}$.*

Proof. **(Classical)** We show $a \overset{\text{co}}{\Rightarrow} v$ and $a \not\Rightarrow v$ implies $a \overset{\infty}{\Rightarrow}$. By coinduction, it suffices to show consistency with $a \overset{\infty}{\Rightarrow}$.

This can in turn be proven by induction on a. The base cases for a being a constant, variable or lambda abstraction do not apply because of the hypotheses, so the claim holds vacuously.

So $a = a_1\, a_2$ for some a_1 and a_2. Inversion on $a_1\, a_2 \overset{\text{co}}{\Rightarrow} v$ tells us that $a_1 \overset{\text{co}}{\Rightarrow} \lambda x.\, b$, $a_2 \overset{\text{co}}{\Rightarrow} v'$, and $[v'/x]b \overset{\text{co}}{\Rightarrow} v$ for some b and value v'.

If $a_1 \Rightarrow \lambda x.\, b$, and $a_2 \Rightarrow v'$, and $[v'/x]b \Rightarrow v$, then $\Rightarrow$-app shows $a_1\, a_2 \Rightarrow v$, a contradiction. Hence one of them must be false (classical).

If it is the first, we have consistency with $\overset{\infty}{\Rightarrow}$-app-l. If it is the second, we have consistency with $\overset{\infty}{\Rightarrow}$-app-r. If it is the third, we have consistency with $\overset{\infty}{\Rightarrow}$-app-f. □

However, $\overset{\text{co}}{\Rightarrow}$ captures divergence in a stricter way than $\overset{\infty}{\Rightarrow}$: $\omega\,(0\,0) \overset{\infty}{\Rightarrow}$ because of ω, but $\omega\,(0\,0) \overset{\text{co}}{\not\Rightarrow} v$ for any v. In other words, the converse implication for Proposition 3.31 is not true.

We can also compare $a \overset{\text{co}}{\Rightarrow} v$ with $a \xrightarrow{\text{co}*} v$, like how we do between $a \Rightarrow v$ and $a \to^* v$.

Proposition 3.32. *For all a and value v, $a \overset{\text{co}}{\Rightarrow} v$ implies $a \xrightarrow{\text{co}*} v$.*

Proof. By coinduction, it suffices to show $a \overset{\text{co}}{\Rightarrow} v$ is consistent with $a \xrightarrow{\text{co}*} v$. We take cases on a.

Case: a is a constant. Then only $a \overset{\text{co}}{\Rightarrow} a$ is possible, using $\overset{\text{co}}{\Rightarrow}$-const. This case is consistent with $\xrightarrow{\text{co}*}$-refl.

Case: a is a variable. Impossible since $a \overset{\text{co}}{\Rightarrow} v$.

Case: a is a lambda abstraction. Only $a \overset{\text{co}}{\Rightarrow} a$ is possible, using $\overset{\text{co}}{\Rightarrow}$-fun. This case is consistent with $\xrightarrow{\text{co}*}$-refl.

Case: $a = a_1\,a_2$ for some a_1, a_2. Only $\overset{\text{co}}{\Rightarrow}$-app applies, and we must have $a_1 \overset{\text{co}}{\Rightarrow} \lambda x.\, b$, $a_2 \overset{\text{co}}{\Rightarrow} v'$, and $[v'/x]b \overset{\text{co}}{\Rightarrow} v$ for some b and value v'. By Proposition 3.30, either $a_1 = \lambda x.\, b$ or $a_1 \to a_1'$ and $a_1' \overset{\text{co}}{\Rightarrow} \lambda x.\, b$ for some a_1'. If it is the second case, $\overset{\text{co}}{\Rightarrow}$-app implies $a_1'\,a_2 \overset{\text{co}}{\Rightarrow} v$. By $\to$-app-l, $a_1\,a_2 \to a_1'\,a_2$, so we have consistency with $\xrightarrow{\text{co}*}$-trans.

For the first case, apply the induction hypothesis on a_2 instead. By Proposition 3.30, either $a_2 = v'$ or $a_2 \to a_2'$ and $a_2' \overset{\text{co}}{\Rightarrow} v'$, for some a_2'. For the latter, apply $\overset{\text{co}}{\Rightarrow}$-app again to conclude $a_1\,a_2' \overset{\text{co}}{\Rightarrow} v$. Since a_1 is a value $a_1\,a_2 \to a_1\,a_2'$ by $\to$-app-r, so we again have consistency with $\xrightarrow{\text{co}*}$-trans.

Hence we are left with $a_1 = \lambda x.\, b$ and $a_2 = v'$. Then $a_1\,a_2 \to [v'/x]b$, and we have consistency with $\xrightarrow{\text{co}*}$-trans. □

Unfortunately, the converse implication is again false, as witnessed by the same example $\omega\,(0\,0)$.

4 Conclusion

In this project, we have investigated coinduction and its use in defining divergence semantics. We have also shown how coinduction can be used to reason about them, in particular with respect to equivalences between variants.

References

[1] Patrick Cousot and Radhia Cousot. Inductive definitions, semantics and abstract interpretations. In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '92, pages 83–94, New York, NY, USA, 1992. ACM.

[2] G. Kahn. Natural semantics. In *4th Annual Symposium on Theoretical Aspects of Computer Sciences on STACS 87*, pages 22–39, London, UK, UK, 1987. Springer-Verlag.

[3] Xavier Leroy. Coinductive big-step operational semantics. In *European Symposium on Programming (ESOP 2006*, pages 54–68. Springer, 2006.

[4] David Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, 1981.

[5] G. D. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University (1981), 1981.

[6] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115:38–94, 1992.