Equality of Corecursive Streams Defined by Finitary Equational Systems

Davide Ancona[1], Pietro Barbieri[1,*] and Elena Zucca[1]

[1]*University of Genova, Italy*

Abstract

In recent work, non-regular streams have been defined corecursively, by representing them with finitary equational systems built on top of various operators, besides the standard constructor. With finitary equational systems based only on the stream constructor, one can use the free theory of regular (a.k.a. rational) trees to get a sound and complete procedure to decide whether two streams are equal. However, this is not the case if one allows other operators in equations, since the underlying equational theory becomes non-trivial, hence equality of regular trees is too strong to guarantee termination of corecursive functions defined even only with the constructor and tail operators. To overcome this problem, we provide a weaker definition of equality between streams denoted by finitary equational systems built on different stream operators, including tail operator and constructor, and prove its soundness.

Keywords

operational semantics, stream programming, regular trees

1. Introduction

In recent work [1, 2], we proposed a novel calculus of numeric streams based on the following key features:

- Streams (infinite sequences) are represented with finitary equational systems built on top of various operators, besides the standard constructor, including tail, interleaving and pointwise binary arithmetic operators. Such systems are modeled as *environments* mapping (finite sets of) stream variables to terms built on variables and the aforementioned operators. For instance, the environment $\rho = \{x \mapsto 1\!:\!2\!:\!x, y \mapsto 1\!:\!y, z \mapsto x[+]y\}$, where $:$ is the stream constructor and $[+]$ the pointwise addition, defines three infinite streams, associated with x, y and z, respectively, which are the unique solution of the system, namely, the stream alternating 1 and 2, the stream repeating 1, and that obtained by pointwise addition of the formers, that is, alternating 2 and 3.
- Such equational systems are defined by recursive functions, such as

```
one_two() = 1:2:one_two()
repeat(n) = n:repeat(n)
incr(s) = s [+] repeat(1)
```

- Function definitions do not have the standard inductive semantics. That is, their calls, rather than leading to non-termination, return pairs (x, ρ) where x is a variable and ρ an *environment*, representing an equational system. For instance:

  `one_two()` evaluates to $(x, \{x \mapsto 1:2:x\})$
  `repeat(1)` evaluates to $(y, \{y \mapsto 1:y\})$
  `incr(one_two())` evaluates to $(z, x \mapsto 1:2:x, y \mapsto 1:y, z \mapsto x[+]y\})$

  This is achieved, despite the evaluation strategy is call-by-value, thanks to the fact that *cyclic calls are detected* and *stream operators are not evaluated.*

This mechanism generalizes *regular corecursion* [3, 4, 5, 6], where the only stream operator which can be used to build equational systems is the constructor, as in $x \mapsto 1:2:x$. In this case, one can show that, under suitable assumptions on the equations, the solutions of the systems correspond to the free theory of *rational* (a.k.a. *regular*) trees [7, 8], that is, finitely branching trees with possibly infinite depth, but only finitely many subtrees; more concretely, these correspond to possibly cyclic terms. Hence the only streams which can be represented in this way are those cyclic, also called regular. Instead, in our generalized approach where other operators can appear in the equational systems, some non-regular streams can be represented as well, as shown by the function `nat` below which returns the stream of natural numbers, represented by $(x, \{x \mapsto 0:(x[+]y), y \mapsto 1:y\})$.

```
nat() = 0:(nat()[+] repeat(1))
```

Such an augmented expressive power comes at some price: (1) conditions on equational systems are needed to ensure unique solutions and (2) equality on streams becomes more complex. While previous work [1, 2] addressed issue (1), this paper focuses on the problem of equality between non-regular streams. Indeed, for simplicity, in [1, 2] we considered syntactic equality, which is trivially sound and decidable, but fails to identify, e.g., $(x, \{x \mapsto 1:x\})$ and $(y, \{y \mapsto 1:1:y\})$, which denote the same stream. The streams that are solutions of these two equations can be proved to be equal if one considers the corresponding regular trees; indeed, for regular trees a decidable sound and complete procedure exists[1] to decide equality.

However, while the theory of regular trees is free, when one allows in the equational systems operators which are not constructors, the underlying equational theory becomes non-trivial and, hence, equality of regular trees is too strong; for instance, it fails to identify $(x, \{x \mapsto 1:x\})$ and $(y, \{x \mapsto 1:x, y \mapsto x\hat{}\})$ where $\_\hat{}$ is the tail operator. This has a negative impact on cycle detection, which is based on equality of function calls to avoid non termination: if a suitable notion of equality is not adopted, then the augmented expressive power achieved with finite representations of non-regular streams is partially lost. Indeed, some function calls may not terminate since cycle detection fails.

After recalling the calculus in Sect. 2, in Sect. 3 we provide our definition of equality and prove its soundness, in the sense that getting the i-th element of provably equal stream expressions gives the same result. In Sect. 4 we show examples of equality checks, Sect. 5 provides an alternative definition of equality, which ensures termination in the positive case by a cycle detection mechanism, and in Sect. 6 we discuss further work.

---

[1]See for instance, the support for cyclic terms in SWI-Prolog https://www.swi-prolog.org/pldoc/man?section=cyclic

2. Stream Calculus

In this section we present the calculus and discuss its features. Fig. 1 shows the syntax.

$\overline{fd}$	$::=$	$fd_1 \ldots fd_n$	program
fd	$::=$	$f(\overline{x}) = se$	function declaration
e	$::=$	$se \mid ne \mid be$	expression
se	$::=$	$x \mid \text{if } be \text{ then } se_1 \text{ else } se_2 \mid ne\!:\!se \mid se\hat{\ } \mid se_1 \; op \; se_2 \mid f(\overline{e})$	stream expression
ne	$::=$	$x \mid se(ne) \mid ne_1 \; op \; ne_2 \mid 0 \mid 1 \mid 2 \mid \ldots$	numeric expression
be	$::=$	$x \mid \text{true} \mid \text{false} \mid \ldots$	boolean expression
op	$::=$	$[nop] \mid \parallel$	binary stream operator
nop	$::=$	$+ \mid - \mid * \mid /$	arithmetic operation

Figure 1: Stream calculus: syntax

A program is a sequence of (mutually recursive) function declarations, for simplicity assumed to only return streams. Stream expressions are variables, conditional expressions, expressions built by stream operators, and function calls. We consider the following stream operators: constructor (prepending a numeric element), tail (denoted by a caret), pointwise arithmetic operations and the interleaving operator ($\parallel$), giving a stream whose elements are alternatively those of the arguments. This latter operator is interesting because it cannot be derived from the others; indeed, implementing the interleaving with just a recursive call involving the first element and the tail of a stream would only work for cyclic streams. Numeric expressions include the access to the i-th[2] element of a stream. We use $\overline{fd}$ to denote a sequence $fd_1, \ldots, fd_n$, with $n \geq 0$, of function declarations, and analogously for other sequences.[3]

The operational semantics, given in Fig. 2, is based on two key ideas:

1. streams are represented with finitary equational systems built with the aforementioned stream operators
2. evaluation keeps track of already considered calls to allow cycle detection

To obtain point (1), an equational system is modeled by an *environment* ρ mapping a finite set of variables into *(open) stream values s*, built on top of stream variables, numeric values and the stream operators; consequently, the *result* of the evaluation of a stream expression is a pair (s, ρ), similarly as done with *capsules* [9] to support cyclic references. Indeed, since a system of equations defines a tuple of streams, we need to specify a value that identifies a specific stream. For instance, $(x, \{x \mapsto n\!:\!x\})$ denotes the stream constantly equal to *n*.

To obtain point (2), evaluation has an additional parameter which is a *call trace*, a map from function calls where arguments have been evaluated, that is, of shape $f(\overline{v})$ (*calls* for short), into variables.

Altogether, the semantic judgment has shape $e, \rho, \tau \Downarrow (v, \rho')$, where *e* is the expression to be evaluated, ρ the current environment defining variables that can occur in *e*, τ the call trace, and (v, ρ') the result. The semantic judgments should be indexed by an underlying (fixed) program, omitted for sake of simplicity. Rules use the following auxiliary definitions:

---

[2]For simplicity, here indexing and numeric expressions coincide, even though indexes are expected to be natural numbers, while values in streams can range over a larger numeric domain.

[3]Hence, declarations and calls of constant functions are included.

$$
\begin{array}{lll}
c & ::= & f(\bar{v}) & \text{(evaluated) call} \\
v & ::= & s \mid n \mid b & \text{value} \\
s & ::= & x \mid n{:}s \mid s\hat{} \mid s_1 \; op \; s_2 & \text{(open) stream value} \\
i, n & ::= & 0 \mid 1 \mid 2 \mid \ldots & \text{index, numeric value} \\
b & ::= & \texttt{true} \mid \texttt{false} & \text{boolean value} \\
\tau & ::= & c_1 \mapsto x_1 \; \ldots \; c_n \mapsto x_n \quad (n \geq 0) & \text{call trace} \\
\rho & ::= & \{x_1 \mapsto s_1 \ldots x_n \mapsto s_n\} \quad (n \geq 0) & \text{environment}
\end{array}
$$

$$\text{(VAL)} \; \frac{}{v, \rho, \tau \Downarrow (v, \rho)}$$

$$\text{(IF-T)} \; \frac{be, \rho, \tau \Downarrow (\texttt{true}, \rho) \quad se_1, \rho, \tau \Downarrow (s, \rho')}{\texttt{if } be \texttt{ then } se_1 \texttt{ else } se_2, \rho, \tau \Downarrow (s, \rho')} \qquad \text{(IF-F)} \; \frac{be, \rho, \tau \Downarrow (\texttt{false}, \rho) \quad se_2, \rho, \tau \Downarrow (s, \rho')}{\texttt{if } be \texttt{ then } se_1 \texttt{ else } se_2, \rho, \tau \Downarrow (s, \rho')}$$

$$\text{(CONS)} \; \frac{ne, \rho, \tau \Downarrow (n, \rho) \quad se, \rho, \tau \Downarrow (s, \rho')}{ne{:}se, \rho, \tau \Downarrow (n{:}s, \rho')} \qquad \text{(TAIL)} \; \frac{se, \rho, \tau \Downarrow (s, \rho')}{se\hat{}, \rho, \tau \Downarrow (s\hat{}, \rho')}$$

$$\text{(OP)} \; \frac{se_1, \rho, \tau \Downarrow (s_1, \rho_1) \quad se_2, \rho, \tau \Downarrow (s_2, \rho_2)}{se_1 \; op \; se_2, \rho, \tau \Downarrow (s_1 \; op \; s_2, \rho_1 \sqcup \rho_2)}$$

$$\text{(ARGS)} \; \frac{e_i, \rho, \tau \Downarrow (v_i, \rho_i) \quad \forall i \in 1..n \quad f(\bar{v}), \widehat{\rho}, \tau \Downarrow (s, \rho')}{f(\bar{e}), \rho, \tau \Downarrow (s, \rho')} \quad \begin{array}{l} \bar{e} = e_1, \ldots, e_n \text{ not of shape } \bar{v} \\ \bar{v} = v_1, \ldots, v_n \\ \widehat{\rho} = \bigsqcup_{i \in 1..n} \rho_i \end{array}$$

$$\text{(INVK)} \; \frac{se[\bar{v}/\bar{x}], \rho, \tau\{c \mapsto x\} \Downarrow (s, \rho')}{c, \rho, \tau \Downarrow (x, \rho'\{x \mapsto s\})} \quad \begin{array}{l} c \notin dom(\tau_{\approx\rho}) \\ x \text{ fresh} \\ fbody(f) = (\bar{x}, se) \end{array} \qquad \text{(COREC)} \; \frac{}{c, \rho, \tau \Downarrow (x, \rho)} \quad \tau_{\approx\rho}(c) = x$$

$$\text{(AT)} \; \frac{se, \rho, \tau \Downarrow (s, \rho') \quad ne, \rho, \tau \Downarrow (i, \rho)}{se(ne), \rho, \tau \Downarrow (n, \rho)} \quad at_{\rho'}(s, i){=}n$$

$$\text{(AT-VAR)} \; \frac{at_\rho(\rho(x), i){=}n'}{at_\rho(x, i){=}n'} \qquad \text{(AT-CONS-0)} \; \frac{}{at_\rho(n{:}s, 0){=}n} \qquad \text{(AT-CONS-SUCC)} \; \frac{at_\rho(s, i-1){=}n'}{at_\rho(n{:}s, i){=}n'} \quad i > 0$$

$$\text{(AT-TAIL)} \; \frac{at_\rho(s, i+1){=}n}{at_\rho(s\hat{}, i){=}n} \qquad \text{(AT-NOP)} \; \frac{at_\rho(s_1, i){=}n_1 \quad at_\rho(s_2, i){=}n_2}{at_\rho(s_1 \; [nop] \; s_2, i){=}n_1 \; nop \; n_2}$$

$$\text{(AT-}\|\text{-EVEN)} \; \frac{at_\rho(s_1, i){=}n}{at_\rho(s_1 \parallel s_2, 2i){=}n} \qquad \text{(AT-}\|\text{-ODD)} \; \frac{at_\rho(s_2, i){=}n}{at_\rho(s_1 \parallel s_2, 2i+1){=}n}$$

Figure 2: Stream calculus: operational semantics

- $\rho \sqcup \rho'$ is the union of two environments, which is well-defined if they have disjoint domains; $\rho\{x \mapsto s\}$ is the environment which gives s on x, coincides with ρ elsewhere; we use analogous notations for call traces.
- $se[\bar{v}/\bar{x}]$ is obtained by parallel substitution of variables $\bar{x}$ with values $\bar{v}$.
- $fbody(f)$ returns the pair of the parameters and the body of the declaration of f, if any, in the assumed program.

Moreover, rules for calls depend on an *equality judgment* $v_1 \approx_\rho v_2$, which will be defined in Sect. 3, stating that v_1 and v_2 are considered equal in the environment ρ. Indeed, look-up of a call in the the call trace is performed *modulo equality*, that is:

- $c \approx_\rho c'$ iff $c = f(v_1, \ldots, v_n)$, $c' = f(v'_1, \ldots, v'_n)$ and $v_i \approx_\rho v'_i$ for all $i \in 1..n$
- $\tau_{\approx\rho}(c) = x$ if $\tau(c') = x$, $c' \approx_\rho c$ for some c'
- hence, $c \notin dom(\tau_{\approx\rho})$ if there is no $c' \in dom(\tau)$ s.t. $c \approx_\rho c'$.

Rules for values and conditional are straightforward. In rules (CONS), (TAIL) and (OP), arguments are evaluated, while the stream operator is applied without any further evaluation.

The rules for function call use a mechanism of cycle detection, similar to that in [6]. They are given in a modular way. That is, evaluation of arguments is handled by a separate rule (ARGS).

Rule (INVK) is applied when a call is considered for the first time, as expressed by the first side condition. The body is retrieved by the auxiliary function *fbody*, and evaluated in a call trace where the call has been mapped into a fresh variable.

Rule (COREC) is applied when a call is considered for the second time, as expressed by the side condition $\tau_{\approx\rho}(c) = x$, which means that, in the call trace, a variable was already associated with some c' considered equivalent to c; indeed, as already explained, cycle detection takes place up to equality in the environment. At this point, the variable x is returned as result. However, there is no associated value in the environment yet; in other words, the result (x, ρ) is open at this point. This means that x is undefined until the environment is updated with the corresponding value in rule (INVK). However, x can be safely used as long as the evaluation does not require x to be inspected; e.g., x can be safely passed as an argument to a function call.

For instance, if we consider $\mathtt{f()=g()}$ $\mathtt{g()=1:f()}$, then the judgment $\mathtt{f()}, \emptyset, \emptyset \Downarrow (x, \rho)$, with $\rho = \{x \mapsto y, y \mapsto 1:x\}$, is derivable; however, while the final result (x, ρ) is closed, the derivation contains also judgments with open results, as, e.g., $\mathtt{f()}, \emptyset, \{\mathtt{f()} \mapsto x, \mathtt{g()} \mapsto y\} \Downarrow (x, \emptyset)$ and $\mathtt{g()}, \emptyset, \{\mathtt{f()} \mapsto x\} \Downarrow (y, \{y \mapsto 1:x\})$.

Finally, rule (AT) computes the i-th element of a stream expression. After evaluation of the arguments, the numeric result is obtained by the auxiliary judgment $at_\rho(s, i) = n$, inductively defined at the bottom of the figure.

If the stream value is a variable, rule (AT-VAR), then the evaluation is propagated to the associated stream value in the environment, if any. If, instead, the variable is free in the environment, then the execution is stuck; an implementation should raise a runtime error instead. Rules (AT-∥-EVEN) and (AT-∥-ODD) handle the interleaving operator. The first one is used for even indexes, and propagates the evaluation to the left-hand side stream; analogously, for odd indexes, the second rule is applied and the evaluation is propagated to the right-hand side stream. The remaining rules are self-explanatory; for examples of derivations in this calculus, we point the reader to [1, 2].

3. Equality of Streams

In this section, we consider the problem of the equality of values. As discussed in the Introduction, and shown in the operational semantics in Fig. 2, this issue is relevant not only to provide an equality operator which can be used by the programmer, but also for cycle detection in calls. To illustrate the second issue we show an example.

```
ones() = 1:ones()
incr_reg(s) = (s(0)+1) : incr_reg(s^)
```

Intuitively, the result of `incr_reg(ones())` should be the stream consisting of infinite occurrences of number 2. However, it is easy to see that this is not the case if cycle detection is based on mere syntactic equality. Indeed, if `incr_reg` is called on `ones()`, that is, on the result $(x, \{x \mapsto 1:x\})$, then `incr_reg` is recursively called on $(x\hat{\ }, \{x \mapsto 1:x\})$ and cycle detection fails because x and $x\hat{\ }$ are not syntactically equal, despite they denote the same stream. This leads to non-termination, since rule (COREC) will never be applied.

Again as anticipated in the Introduction, the first step towards a more expressive definition is to consider equality in the free theory of regular terms, as in [6]. This can be done by a coinductive definition[4] which computes the unfolding of variables by looking up their associated values in the environment. This allows us to identify results returned by `ones()` and `altOnes()`, where `altOnes() = 1:1:altOnes()`. Indeed, in the environment of shape $\{x \mapsto 1:x, y \mapsto 1:1:y\}$ resulting from the evaluation, one can check by unfolding that the values associated with x and y correspond to the same regular term. However, if one allows other operators in the equational systems, then the equational theory is no longer free, hence equality of regular terms fails to identify the results of `ones()` and `ones()^` as in the example above.

In order to deal with the tail operator, our solution is based on the key idea that the equality check performs a partial symbolic evaluation of the tail. For instance, with this solution the example `incr_reg(ones())` is correctly handled, since the symbolic evaluation of the tail of $(x, \{x \mapsto 1:x\})$ returns $(x, \{x \mapsto 1:x\})$.

The equality check is formalized by the judgment $s_1 \approx_\rho s_2$, coinductively defined in Fig. 3. In rules (VAR-L) and (VAR-R), a variable defined in the environment is equal to a stream value if the same holds for its associated stream value. In rule (VAR), a variable is equal to itself. In rule (CONS), prepending the same element to equal streams gives equal streams. Analogously, in rule (TAIL-TAIL), the tails of two equal streams are equal. In rule (OP), two streams defined using the same binary operation are equal if their arguments are respectively equal.

The most interesting rules are those handling the case when one of the two sides of the equality is of shape $s\hat{\ }$. Indeed, in such cases an attempt is made at computing (a stream value equal to) the tail of s, through the auxiliary judgment $\mathsf{Tail}_\rho(s)=s'$ defined at the bottom of the figure.[5] Function Tail_ρ is inductively defined, and the base case is the constructor, rule (TAIL-CONS), where the result is simply the tail stream. In the other cases, the function is propagated as expected. In particular, in rule (TAIL), to obtain the result we need two subsequent applications of Tail_ρ, the former getting the tail of the argument. Note also, in rule (∥), that, given a stream

---

[4]As further discussed at the end of this section, for regular terms the coinductive definition can be turned into an equivalent inductive and algorithmic one.

[5]Note the difference between this judgment and the notation $s\hat{\ }$ which is part of the syntax of streams (Fig. 1).

$$
\begin{array}{lll}
s & ::= & x \mid n\!:\!s \mid s\hat{} \mid s_1 \; op \; s_2 \qquad \text{(open) stream value} \\
op & ::= & [nop] \mid \| \qquad\qquad\qquad\quad \text{binary stream operator} \\
\rho & ::= & x_1 \mapsto s_1 \ldots x_n \mapsto s_n \quad (n \geq 0) \quad \text{environment}
\end{array}
$$

$$
\text{(VAR-L)} \frac{\rho(x) \approx_\rho s}{x \approx_\rho s}
\qquad
\text{(VAR-R)} \frac{s \approx_\rho \rho(x)}{s \approx_\rho x}
\qquad
\text{(VAR)} \frac{}{x \approx_\rho x}
\qquad
\text{(CONS)} \frac{s_1 \approx_\rho s_2}{n\!:\!s_1 \approx_\rho n\!:\!s_2}
\qquad
\text{(TAIL-TAIL)} \frac{s_1 \approx_\rho s_2}{s_1\hat{} \approx_\rho s_2\hat{}}
$$

$$
\text{(OP)} \frac{s_1 \approx_\rho s_1' \quad s_2 \approx_\rho s_2'}{s_1 \; op \; s_2 \approx_\rho s_1' \; op \; s_2'}
\qquad
\text{(TAIL-L)} \frac{s' \approx_\rho s_2 \quad \mathsf{Tail}_\rho(s_1) = s'}{s_1\hat{} \approx_\rho s_2}
\qquad
\text{(TAIL-R)} \frac{s_1 \approx_\rho s' \quad \mathsf{Tail}_\rho(s_2) = s'}{s_1 \approx_\rho s_2\hat{}}
$$

$$
\text{(TAIL-CONS)} \frac{}{\mathsf{Tail}_\rho(n\!:\!s) = s}
\qquad
\text{(VAR)} \frac{\mathsf{Tail}_\rho(\rho(x)) = s}{\mathsf{Tail}_\rho(x) = s}
\qquad
\text{(TAIL)} \frac{\mathsf{Tail}_\rho(s) = s' \quad \mathsf{Tail}_\rho(s') = s''}{\mathsf{Tail}_\rho(s\hat{}) = s''}
$$

$$
\text{(NOP)} \frac{\mathsf{Tail}_\rho(s_1) = s_1' \quad \mathsf{Tail}_\rho(s_2) = s_2'}{\mathsf{Tail}_\rho(s_1[nop]s_2) = s_1'[nop]s_2'}
\qquad
\text{(\|)} \frac{\mathsf{Tail}_\rho(s_1) = s_1'}{\mathsf{Tail}_\rho(s_1 \| s_2) = s_2 \| s_1'}
$$

Figure 3: Equality check

whose elements are alternatively those of the arguments, the elements of the tail are alternatively those of the second argument and the tail of the first.

To explain what does it mean that the definition of the judgment $s_1 \approx_\rho s_2$ in Fig. 3 is coinductive, we briefly recall some notions on inference systems [10, 11, 12]. An *inference system* is a set of *rules* of shape $\frac{j_1,\ldots,j_n}{j}$ where the *premises* $j_1, \ldots, j_n$ and the consequence j are elements of a given set of *judgments*. A *proof tree* for a judgment j is a tree whose nodes are (labelled with) judgments where the root is (labelled with) j and, for each node (labeled) j' and (labels of the) children $j_1, \ldots, j_n$, there is a rule $\frac{j_1,\ldots,j_n}{j'}$. Then, the set of the judgments *inductively defined* are those with a *finite* proof tree, whereas the set of judgments *coinductively defined* are those with an either finite or infinite proof tree. In our case, the coinductive definition is the most natural and abstract for infinite objects such as streams, and is convenient for the soundness proof. It is possible to provide an alternative inductive definition by following the standard technique, adopted in co-SLD resolution [13, 14] and proved sound for an arbitrary inference system in [5], of adding already considered judgments as *coinductive hypotheses*. Such an inductive definition is effective, meaning that there is a procedure which decides whether equality between two regular terms is derivable or not, thanks to the assumption that regular terms can only have a finite set of sub-terms. In our case, this leads to a judgment $\varepsilon \vdash s_1 \approx_\rho s_2$, reported in Sect. 5, which keeps a *trace* ε of the already considered pairs. This inductive definition provides a first step towards an algorithm, as we will discuss in the Conclusion.

We now prove the soundness of the equality check. That is, if we derive that two streams are equal, then they are equal in the sense that access to an arbitrary index will give the same result. Soundness of the equality check (Theorem 3.2) relies on soundness of the Tail judgment (Theorem 3.1). To the aim of the proof of Theorem 3.2, we introduce the notation $at_\rho(s, i+1) \overset{k}{=} n$, meaning that the proof tree of $at_\rho(s, i+1) = n$ has depth k.

Theorem 3.1. *If* $\mathsf{Tail}_\rho(s){=}s'$ *holds, then, for all* $i, k{\geq}0$, $at_\rho(s, i+1)\stackrel{k}{=}n$ *implies* $at_\rho(s', i)\stackrel{k'}{=}n$ *for some* $k'{\leq}k$.

Proof. By induction on the rules defining $\mathsf{Tail}_\rho(s){=}s'$.

(TAIL-CONS) This is an axiom with conclusion $\mathsf{Tail}_\rho(n\!:\!s){=}s$. We have to show that, for all $i{\geq}0$, $at_\rho(n\!:\!s, i+1)\stackrel{k}{=}n$ implies $at_\rho(s, i)\stackrel{k'}{=}n$ with $k'{\leq}k$, and this is true since $at_\rho(n\!:\!s, i+1)\stackrel{k}{=}n$ is necessarily derived from $at_\rho(s, i)\stackrel{k-1}{=}n$ by rule (AT-CONS-SUCC).

(VAR) $\mathsf{Tail}_\rho(\rho(x)){=}s$ is the premise and $\mathsf{Tail}_\rho(x){=}s$ the conclusion. We have to show that, for all $i{\geq}0$, $at_\rho(x, i+1)\stackrel{k}{=}n$ implies $at_\rho(s, i)\stackrel{k'}{=}n$ with $k'{\leq}k$. Since $at_\rho(x, i+1)\stackrel{k}{=}n$ is necessarily derived from $at_\rho(\rho(x), i+1)\stackrel{k-1}{=}n$ by rule (AT-VAR), by inductive hypothesis we have $at_\rho(s, i)\stackrel{k'}{=}n$ with $k'{\leq}k-1{<}k$.

(TAIL) $\mathsf{Tail}_\rho(s){=}s'$, $\mathsf{Tail}_\rho(s'){=}s''$ are the premises, and $\mathsf{Tail}_\rho(s\hat{\,}){=}s''$ the conclusion. We have to show that, for all $i{\geq}0$, $at_\rho(s\hat{\,}, i+1)\stackrel{k}{=}n$ implies $at_\rho(s'', i)\stackrel{k''}{=}n$ with $k''{\leq}k$. Since $at_\rho(s\hat{\,}, i+1)\stackrel{k}{=}n$ is necessarily derived from $at_\rho(s, i+2)\stackrel{k-1}{=}n$ by rule (AT-TAIL), by inductive hypothesis on the first premise we have $at_\rho(s', i+1)\stackrel{k'}{=}n$, with $k'{\leq}k-1{<}k$. Moreover, by inductive hypothesis on the second premise, we have $at_\rho(s'', i)\stackrel{k''}{=}n$, with $k''{\leq}k'{\leq}k-1{<}k$.

(NOP) $\mathsf{Tail}_\rho(s_1){=}s_1'$, $\mathsf{Tail}_\rho(s_2){=}s_2'$ are the premises, and $\mathsf{Tail}_\rho(s_1[nop]s_2){=}s_1'[nop]s_2'$ the conclusion. We have to show that, for all $i{\geq}0$, $at_\rho(s_1[nop]s_2, i+1)\stackrel{k}{=}n$ implies $at_\rho(s_1'[nop]s_2', i)\stackrel{k'}{=}n$ with $k'{\leq}k$. Since $at_\rho(s_1[nop]s_2, i+1)\stackrel{k}{=}n$ is necessarily derived from $at_\rho(s_1, i+1)\stackrel{k-1}{=}n_1$ and $at_\rho(s_2, i+1)\stackrel{k-1}{=}n_2$, with $n = n_1 \; nop \; n_2$, by rule (AT-NOP), by inductive hypothesis on the first and second premise we have $at_\rho(s_1', i)\stackrel{k_1}{=}n_1$ and $at_\rho(s_2', i)\stackrel{k_2}{=}n_2$ with $k_1, k_2{\leq}k-1$, respectively. Hence, by rule (AT-NOP), we have $at_\rho(s_1'[nop]s_2', i)\stackrel{k'}{=}n_1 \; nop \; n_2 = n$, with $k' = \max(k_1, k_2) + 1{\leq}k-1+1 = k$.

($\parallel$) $\mathsf{Tail}_\rho(s_1){=}s_1'$ is the premise and $\mathsf{Tail}_\rho(s_1 \parallel s_2){=}s_2 \parallel s_1'$ the conclusion. We have to show that, for all $i{\geq}0$, $at_\rho(s_1 \parallel s_2, i+1)\stackrel{k}{=}n$ implies $at_\rho(s_2 \parallel s_1', i)\stackrel{k'}{=}n$ with $k'{\leq}k$. The proof proceeds by case analysis on the parity of i:

i **even:** $at_\rho(s_1 \parallel s_2, i+1)\stackrel{k}{=}n$ is necessarily derived from $at_\rho(s_2, \frac{i}{2})\stackrel{k-1}{=}n$ by rule (AT-$\parallel$-ODD); hence, by rule (AT-$\parallel$-EVEN) we have $at_\rho(s_2 \parallel s_1', i)\stackrel{k'}{=}n$ with $k' = k$.

i **odd:** $at_\rho(s_1 \parallel s_2, i+1)\stackrel{k}{=}n$ is necessarily derived from $at_\rho(s_1, \frac{i+1}{2})\stackrel{k-1}{=}n$ by rule (AT-$\parallel$-EVEN); that is, $at_\rho(s_1, \frac{i-1}{2}+1)\stackrel{k-1}{=}n$, hence, by inductive hypothesis, $at_\rho(s_1', \frac{i-1}{2})\stackrel{k''}{=}n$ with $k''{\leq}k-1$. Finally, by rule (AT-$\parallel$-ODD) we have $at_\rho(s_2 \parallel s_1', i)\stackrel{k'}{=}n$ with $k' = k'' + 1{\leq}k$.

$\square$

Theorem 3.2. *For all* $i{\geq}0$, *if* $at_\rho(s, i){=}n$, $at_\rho(s', i){=}n'$, *and* $s \approx_\rho s'$, *then* $n = n'$.

Proof. Assume that $at_\rho(s,i)\overset{k}{=}n$, $at_\rho(s',i)\overset{k'}{=}n'$. The proof is by Noetherian induction on the pairs (k,k'), with the componentwise order, that is, $(k_1,k_2)\le(k'_1,k'_2)$ iff $k_1\le k'_1$ and $k_2\le k'_2$.

Base We consider the pair $(0,0)$, that is, the case when the two judgments are derived by axiom (AT-CONS-0), hence, for $i=0$. We have $at_\rho(n\!:\!s,0)=n$, and $at_\rho(n'\!:\!s',0)=n'$. Moreover, we have $n\!:\!s \approx_\rho n'\!:\!s'$, which has been necessarily derived by rule (CONS), hence $n=n'$.

Inductive step We consider pairs (k,k') where either $k>0$ or $k'>0$, and proceed by case analysis on the (last) rule applied to derive the judgment $s \approx_\rho s'$.

(VAR-L) We have $x \approx_\rho s$ and $\rho(x) \approx_\rho s$. Moreover, we have $at_\rho(x,i)\overset{k}{=}n$, $at_\rho(s,i)\overset{k'}{=}n'$ and, since the former judgment has been necessarily derived by rule (AT-VAR), $at_\rho(\rho(x),i)\overset{k-1}{=}n$. We apply the inductive hypothesis and get the thesis.

(VAR-R) This case is symmetrical to the one above.

(VAR) We have $x \approx_\rho x$. We conclude by the fact that the judgment $at_\rho(s,i)=n$ is deterministic, therefore $at_\rho(x,i)=n_1$ and $at_\rho(x,i)=n_2$ implies $n_1=n_2$.

(CONS) We have $m\!:\!s \approx_\rho m'\!:\!s'$ and $s \approx_\rho s'$. Moreover, we have $at_\rho(m\!:\!s,i)\overset{k}{=}n$, $at_\rho(m'\!:\!s',i)\overset{k'}{=}n'$ and, since $k>0$ or $k'>0$, we have $i>0$ and these judgments have been necessarily derived by rule (AT-CONS-SUCC); therefore, $at_\rho(s,i-1)\overset{k-1}{=}n$, $at_\rho(s',i-1)\overset{k'-1}{=}n'$. We apply the inductive hypothesis and get the thesis.

(TAIL-TAIL) We have $s\hat{\ }\approx_\rho s'\hat{\ }$ and $s\approx_\rho s'$. Moreover, we have $at_\rho(s\hat{\ },i)\overset{k}{=}n$, $at_\rho(s'\hat{\ },i)\overset{k'}{=}n'$ and, since these judgments have been necessarily derived by rule (AT-TAIL), $at_\rho(s,i+1)\overset{k-1}{=}n$, and $at_\rho(s',i+1)\overset{k'-1}{=}n'$. We apply the inductive hypothesis and get the thesis.

(OP) By cases on the binary operator *op*.

(NOP): We have $s_1\ [nop]\ s_2 \approx_\rho s'_1\ [nop]\ s'_2$, $s_1 \approx_\rho s'_1$ and $s_2 \approx_\rho s'_2$. Moreover, we have $at_\rho(s_1\ [nop]\ s_2,i)\overset{k}{=}n$, $at_\rho(s'_1\ [nop]\ s'_2,i)\overset{k'}{=}n'$ and, since these judgments have been necessarily derived by rule (AT-NOP), $at_\rho(s_1,i)\overset{k-1}{=}l$, $at_\rho(s_2,i)\overset{k-1}{=}m$, $at_\rho(s'_1,i)\overset{k'-1}{=}l'$, $at_\rho(s'_2,i)\overset{k'-1}{=}m'$, $n=l\ nop\ m$, $n'=l'\ nop\ m'$. We apply the inductive hypothesis and get the thesis.

(∥): By cases on the parity of i.

i **even:** We have $s_1 \parallel s_2 \approx_\rho s'_1 \parallel s'_2$ and $s_1 \approx_\rho s'_1$. Moreover, we have $at_\rho(s_1 \parallel s_2,i)\overset{k}{=}n$, $at_\rho(s'_1 \parallel s'_2,i)\overset{k'}{=}n'$ and, since these judgments have been necessarily derived by rule (AT-∥-EVEN), $at_\rho(s_1,\frac{i}{2})\overset{k-1}{=}n$, $at_\rho(s'_1,\frac{i}{2})\overset{k'-1}{=}n'$. We apply the inductive hypothesis and get the thesis.

i **odd:** This case is symmetrical to the one above.

(TAIL-R) We have $s_1\approx_\rho s_2\hat{\ }$ and $s_1\approx_\rho s'$, with $\mathsf{Tail}_\rho(s_2)=s'$. Moreover, we have $at_\rho(s_1,i)\overset{k}{=}n$, $at_\rho(s_2\hat{\ },i)\overset{k'}{=}n'$ and, since the latter judgment has been necessarily derived by rule

(AT-TAIL), $at_\rho(s_2, i+1)^{k'-1}_{=} n'$. From Theorem 3.1 we have that $at_\rho(s', i)^{k''}_{=} n'$ with $k'' \le k' - 1 < k'$. We apply the inductive hypothesis and get the thesis.

(TAIL-L) This case is symmetrical to the one above.

$\square$

4. Examples

We illustrate how the equality check works by some examples. The first one is shown in Fig. 4.

$$
\frac{
\dfrac{
\dfrac{\vdots}{x \approx_\rho y\hat{}\hat{}} \text{ (VAR-L)}
}{1 : x \approx_\rho 1 : y\hat{}\hat{}} \text{ (CONS)} \qquad \mathsf{Tail}_\rho(y\hat{}) = 1 : y\hat{}\hat{}
}{
\dfrac{1 : x \approx_\rho y\hat{}\hat{}}{x \approx_\rho y\hat{}\hat{}} \text{ (VAR-L)}
} \text{ (TAIL-R)}
$$

$$
\frac{
\dfrac{
\dfrac{\overline{\mathsf{Tail}_\rho(2 : 3 : 1 : y\hat{}\hat{}) = 3 : 1 : y\hat{}\hat{}}}{\mathsf{Tail}_\rho(y) = 3 : 1 : y\hat{}\hat{}} \text{ (VAR)}
\qquad
\dfrac{\overline{\mathsf{Tail}_\rho(3 : 1 : y\hat{}\hat{}) = 1 : y\hat{}\hat{}}}{} \text{ (CONS)}
}{\mathsf{Tail}_\rho(y\hat{}) = 1 : y\hat{}\hat{}} \text{ (TAIL)}
}{}
$$

Figure 4: $x \approx_\rho y\hat{}\hat{}$ with $\rho = \{x \mapsto 1 : x, y \mapsto 2 : 3 : 1 : y\hat{}\hat{}\}$

In this proof tree, rule (VAR-L) is applied first, and then rule (TAIL-R); we could have applied the rules in the other order as well. The proof tree of $\mathsf{Tail}_\rho(y\hat{}) = 1 : y\hat{}\hat{}$ is shown in the bottom part of the figure. The result is computed by applying twice the tail operator, and unfolding y. Then, in the main proof tree, rule (CONS) is applied since the first element of both streams is 1. It is easy to see that there is a regular infinite proof tree, denoted by the dots.

In the second one, in Fig. 5, we show how the equality check deals with non-regular streams.

$$
\frac{
\dfrac{
\dfrac{
\dfrac{\dfrac{\vdots}{x \approx_\rho y} \text{(VAR-L)} \quad \dfrac{\vdots}{x \approx_\rho y} \text{(VAR-L)}}{x \ [+] \ x \approx_\rho y \ [+] \ y} \text{(OP)} \quad \dfrac{\dfrac{\overline{\mathsf{Tail}_\rho(1 : y) = y}}{} \text{(TAIL-CONS)}}{\mathsf{Tail}_\rho((1 : y) \ [+] \ (1 : y)) = y \ [+] \ y} \text{(TAIL-NOP)}
}{x \ [+] \ x \approx_\rho ((1 : y) \ [+] \ (1 : y))\hat{}} \text{(TAIL-R)}
}{1 : (x \ [+] \ x) \approx_\rho 1 : ((1 : y) \ [+] \ (1 : y))\hat{}} \text{(CONS)}
}{
\dfrac{1 : (x \ [+] \ x) \approx_\rho y}{x \approx_\rho y} \text{(VAR-L)}
} \text{(VAR-R)}
$$

Figure 5: $x \approx_\rho y$ with $\rho = \{x \mapsto 1 : (x \ [+] \ x), y \mapsto 1 : ((1 : y) \ [+] \ (1 : y))\hat{}\}$

Here, both x and y denote the stream of all powers of 2. The derivation starts with the unfolding of both variables, followed by the application of rule (cons). Then, the tail of the second component is computed, with the proof tree shown as additional premise in rule (tail-r). After that, the derivation continues by distributing the equality check to the substreams x and y. It is easy to see that there is a regular infinite proof tree, denoted by the dots.

5. Equality Check with Trace

We provide an alternative definition of equality, which ensures termination in the positive case by a cycle detection mechanism. The judgment $\varepsilon \vdash s_1 \approx_\rho s_2$, defined in Fig. 6, means that s_1 and s_2 are equal in ρ under the *equality trace* ε, containing coinductive hypotheses as pairs of already considered stream values. The trace is abstractly considered as a set, so order and repetitions are immaterial.

$$
\begin{array}{llll}
s & ::= & x \mid n\!:\!s \mid s\,\hat{} \mid s_1\, op\, s_2 & \text{(open) stream value} \\
op & ::= & [nop] \mid | \mid \| & \text{binary stream operator} \\
\rho & ::= & \{x_1 \mapsto s_1 \ldots x_n \mapsto s_n\} \quad (n \geq 0) & \text{environment} \\
\varepsilon & ::= & (s_1, s_1') \ldots (s_n, s_n') \quad (n \geq 0) & \text{equality trace}
\end{array}
$$

$$
\text{(var-l)}\,\frac{\varepsilon \cdot (x, s) \vdash \rho(x) \approx_\rho s}{\varepsilon \vdash x \approx_\rho s}
\qquad
\text{(var-r)}\,\frac{\varepsilon \cdot (s, x) \vdash s \approx_\rho \rho(x)}{\varepsilon \vdash s \approx_\rho x}
\qquad
\text{(var)}\,\frac{}{\varepsilon \vdash x \approx_\rho x}
\qquad
\text{(cons)}\,\frac{\varepsilon \vdash s_1 \approx_\rho s_2}{\varepsilon \vdash n\!:\!s_1 \approx_\rho n\!:\!s_2}
$$

$$
\text{(tail-tail)}\,\frac{\varepsilon \vdash s_1 \approx_\rho s_2}{\varepsilon \vdash s_1\,\hat{} \approx_\rho s_2\,\hat{}}
\qquad
\text{(tail-l)}\,\frac{\varepsilon \vdash s' \approx_\rho s_2}{\varepsilon \vdash s_1\,\hat{} \approx_\rho s_2} \ \ \mathsf{Tail}_\rho(s_1)\!=\!s'
\qquad
\text{(tail-r)}\,\frac{\varepsilon \vdash s_1 \approx_\rho s'}{\varepsilon \vdash s_1 \approx_\rho s_2\,\hat{}} \ \ \mathsf{Tail}_\rho(s_2)\!=\!s'
$$

$$
\text{(op)}\,\frac{\varepsilon \vdash s_1 \approx_\rho s_1' \qquad \varepsilon \vdash s_2 \approx_\rho s_2'}{\varepsilon \vdash s_1\, op\, s_2 \approx_\rho s_1'\, op\, s_2'}
\qquad
\text{(corec)}\,\frac{}{\varepsilon \vdash s_1 \approx_\rho s_2} \ \ (s_1, s_2) \in \varepsilon
$$

Figure 6: Equality of streams with equality trace

The rules are analogous to those in Fig. 3, except that in (var-l) and (var-r) considered pairs are added as coinductive hypotheses in the trace, and rule (corec) states that two stream values are equal if the same pair is contained in the trace of coinductive hypotheses, that is, it has been already considered. The function Tail_ρ is that defined in Fig. 3. Whereas the judgment $s_1 \approx_\rho s_2$ in Fig. 3 is defined coinductively, the judgment $\varepsilon \vdash s_1 \approx_\rho s_2$ is defined inductively. The soundness of $\emptyset \vdash s_1 \approx_\rho s_2$ with respect to $s_1 \approx_\rho s_2$ follows from a general result proved in [5]. As an example, in Fig. 7 we show the derivation of the judgment with the equality trace for the first example of Sect. 4. The proof tree is exactly analogous to that in Fig. 4, with only two differences: a new pair is added to the equality trace ε when rules (var-l) and (var-r) are applied, while rule (corec) is used to end the derivation, as soon as an already processed pair is found.

$$\dfrac{\dfrac{\dfrac{\overline{\{(x,y\char`\^\char`\^)\} \vdash x \approx_\rho y\char`\^\char`\^}}{\{(x,y\char`\^\char`\^)\} \vdash 1 : x \approx_\rho 1 : y\char`\^\char`\^}}{\{(x,y\char`\^\char`\^)\} \vdash 1 : x \approx_\rho y\char`\^\char`\^} \quad \mathrm{Tail}_\rho(y\char`\^)=1:y\char`\^\char`\^}{\emptyset \vdash x \approx_\rho y\char`\^\char`\^}$$

(COREC) (CONS) (TAIL-R) (VAR-L)

Figure 7: $\emptyset \vdash x \approx_\rho y\char`\^\char`\^$ with $\rho = \{x \mapsto 1:x, y \mapsto 2:3:1:y\char`\^\char`\^\}$

6. Conclusion

We provided an operational characterization of equality between streams which goes beyond equality of regular terms, by performing a partial symbolic evaluation of the tail operator.

Such operational characterization has been described by a coinductive inference system, that is, infinite derivations are allowed. This description is very natural and abstract, and convenient for the soundness proof, but clearly non-algorithmic. Thus, the first objective for future work is to provide a sound algorithm to check equality. To this end, the first step, shown in Sect. 5, is to convert the coinductive definition into an inductive one, where derivations are finite. This has been done by the standard technique, adopted in co-SLD resolution [13, 14] and proved sound for an arbitrary inference system in [5], of adding already considered judgments as *coinductive hypotheses*. Thanks to this general result, we know that, if we have a finite derivation for $\emptyset \vdash s_1 \approx_\rho s_2$, then we have a possibly infinite, but regular derivation tree for $s_1 \approx_\rho s_2$ as well, hence by Theorem 3.2 we can conclude that the inductive definition of equality is sound.

The following further steps are required to actually get an algorithm sound and complete w.r.t. the coinductive definition in Fig. 3, that is, a procedure which always terminates and returns a positive answer iff $s_1 \approx_\rho s_2$ is derivable:

1. a proof that there exists a sound and complete algorithm for the judgment $\mathrm{Tail}_\rho(s)=s'$;
2. a proof that derivation trees for $s_1 \approx_\rho s_2$ are all *regular*, that is, the set of checked pairs is *finite*. This is not so simple as happens for equality of regular terms, because checking that $s_1 \approx_\rho s_2$ is derivable needs inspection of not only the sub-terms of s_1 and s_2, but also the possibly new terms generated by the symbolic computation of tail.

Another issue for further research is completeness; indeed, the characterization proposed in this work fails to capture some laws that hold in the equational theory defined by index access. For instance, if $\rho = \{x \mapsto 0 : x\}$, then, for all $i \geq 0$, $at_\rho(x, i)=at_\rho(x \,[+]\, x, i)=at_\rho(x \,\|\, x, i)=0$ holds, while $x \approx_\rho x \,[+]\, x$ and $x \approx_\rho x \,\|\, x$ are not derivable according to the definition in Fig. 3. Hence, a long-term goal is to investigate more expressive definitions of equality, and decidability of equality for (fragments of) the stream calculus.

References

[1] D. Ancona, P. Barbieri, E. Zucca, Enhanced regular corecursion for data streams, in: ICTCS'21 - Italian Conf. on Theoretical Computer Science, 2021.

[2] D. Ancona, P. Barbieri, E. Zucca, Checked corecursive streams: trading expressive power for reliability, in: Functional and Logic Programming (FLOPS 2022), 2022.

[3] J.-B. Jeannin, D. Kozen, A. Silva, CoCaml: Functional programming with regular coinductive types, Fundamenta Informaticae 150 (2017) 347–377.

[4] F. Dagnino, D. Ancona, E. Zucca, Flexible coinductive logic programming, Theory and Practice of Logic Programming 20 (2020) 818–833. doi:10.1017/S147106842000023X, issue for ICLP 2020.

[5] F. Dagnino, Foundations of regular coinduction, Logical Methods in Computer Science 17 (2021). doi:10.46298/lmcs-17(4:2)2021.

[6] D. Ancona, P. Barbieri, F. Dagnino, E. Zucca, Sound regular corecursion in coFJ, in: R. Hirschfeld, T. Pape (Eds.), ECOOP'20 - Object-Oriented Programming, volume 166 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 1:1–1:28. doi:10.4230/LIPIcs.ECOOP.2020.1.

[7] B. Courcelle, Fundamental properties of infinite trees, Theoretical Computer Science 25 (1983) 95–169. doi:10.1016/0304-3975(83)90059-2.

[8] J. Adámek, S. Milius, J. Velebil, Free iterative theories: A coalgebraic view, Math. Struct. Comput. Sci. 13 (2003) 259–320. doi:10.1017/S0960129502003924.

[9] J. Jeannin, D. Kozen, Computing with capsules, Journal of Automata, Languages and Combinatorics 17 (2012) 185–204. doi:10.25596/jalc-2012-185.

[10] P. Aczel, An introduction to inductive definitions, in: J. Barwise (Ed.), Handbook of Mathematical Logic, volume 90 of *Studies in Logic and the Foundations of Mathematics*, Elsevier, 1977, pp. 739 – 782.

[11] X. Leroy, H. Grall, Coinductive big-step operational semantics, Information and Computation 207 (2009) 284–304. doi:10.1016/j.ic.2007.12.004.

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

[13] L. Simon, Extending logic programming with coinduction, Ph.D. thesis, University of Texas at Dallas, 2006.

[14] L. Simon, A. Bansal, A. Mallya, G. Gupta, Co-logic programming: Extending logic programming with coinduction, in: L. Arge, C. Cachin, T. Jurdzinski, A. Tarlecki (Eds.), Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, volume 4596 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 472–483. doi:10.1007/978-3-540-73420-8_42.