Theoretical Computer Science 1009 (2024) 114680

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Checking equivalence of corecursive streams: An inductive procedure ☆

Davide Ancona *, Pietro Barbieri, Elena Zucca

DIBRIS, Università di Genova, Via Dodecaneso 35, Genova, Italy

ARTICLE INFO

Keywords:
Operational semantics
Stream programming
Regular trees

ABSTRACT

In recent work, non-periodic streams have been defined corecursively, by representing them with finitary equational systems built on top of various operators, besides the standard constructor. When only the stream constructor is allowed in equations, only periodic streams can be represented, and the structures of periodic streams and infinite regular trees are isomorphic. Therefore, one can use the theory of regular trees to get a sound and complete procedure to decide whether two equational systems are equivalent, that is, define the same streams. However, such an isomorphism no longer exists if one allows other operators in equations; in particular, there exist systems of equations which have the same unique solution as streams, but not as regular trees. Hence, equality of regular trees becomes stronger then equality of streams, with a negative impact on termination of functions whose definition is based on the equivalence of the representation of streams as finitary equational systems. To overcome this problem, we provide a weaker definition of equivalence, and prove its soundness and relative completeness. This definition is coinductive, hence non-algorithmic. However, we show that it can be turned into an equivalent inductive procedure.

1. Introduction

Reference calculus. In recent work [3,4,6], we proposed a novel calculus of numeric streams based on the following key features:

- Streams are functions over natural numbers [8,6], operationally represented by means of finitary equational systems built on top of various operators, including, besides the standard constructor, tail, pointwise binary arithmetic operators, and interleaving. Such systems are modeled as *environments*, that is, (finite) mappings, written $\{x_1 \mapsto s_1 \dots x_n \mapsto s_n\}$, from stream variables x_i to terms s_i built on variables and the aforementioned operators. An example is 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. For a given environment, a *solution* is a mapping from variables into streams (functions over natural numbers) so that the corresponding equational system is satisfied. For instance, it is easy to see that ρ has a unique solution, which associates with x, y and z, respectively, the streams s_x, s_y and s_z such that $s_x(2i) = 1$, $s_x(2i+1) = 2$, $s_y(i) = 1$ and s_z is the pointwise addition of the formers, that is, $s_z(2i) = 2$, $s_z(2i+1) = 3$, for all natural numbers i.

☆ This article belongs to Section B: Logic, semantics and theory of programming, Edited by Don Sannella.

* Corresponding author.
E-mail addresses: davide.ancona@unige.it (D. Ancona), pietro.barbieri@edu.unige.it (P. Barbieri), elena.zucca@unige.it (E. Zucca).

https://doi.org/10.1016/j.tcs.2024.114680
Received 16 January 2023; Received in revised form 29 March 2024; Accepted 4 June 2024

Available online 12 June 2024

- In the operational semantics, stream expressions reduce to pairs of the form (s, ρ). We will call such pairs *operational streams*, since they represent stream values in the operational semantics, as opposed to *abstract streams* which are, as mentioned above, functions over natural numbers, omitting the qualifier when there is no ambiguity.
- A program is a sequence of mutually recursive function definitions, like for instance

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

- Function definitions do not have the standard inductive semantics. That is, some calls, which would lead to non-termination with such semantics, return, instead, streams of the form (x, ρ), despite the evaluation strategy is call-by-value. This is achieved thanks to the fact that *all stream operators are managed as constructors, and, hence, are not evaluated*, and semantics is equipped with *cycle detection*. This means that evaluation keeps track of the pending calls, so to recognize when one of such calls needs to be executed again. For instance, the call `repeat(1)` does not diverge because in the body of the function the same call is detected and, hence, its returned value is associated with the same variable x corresponding to the returned value of the initial call. Therefore, the final result is $(x, \{x \mapsto 1 : x\})$. Similarly,

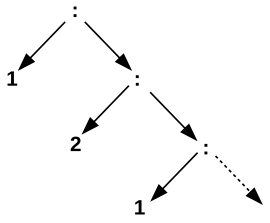
`one_two()` evaluates to $(y, \{y \mapsto 1 : 2 : y\})$
`incr(one_two())` evaluates to $(z, \{x \mapsto 1 : x, y \mapsto 1 : 2 : y, z \mapsto y\,[+]\,x\})$

Regular corecursion. This mechanism generalizes *regular corecursion* [19,12,11,2], where the only stream operator allowed in environments is the constructor with its left operand restricted to numeric constants, as in $x \mapsto 1 : 2 : x$ and $y \mapsto 1 : y$.

In this case, one can show that, if recursive definitions are guarded by the constructor, then there exists a unique solution (up to isomorphism) in the theory of infinite *regular* trees [10,9,1] where inner nodes and leaves are labeled by the constructor and numeric constants, respectively. Infinite *regular* trees are finitely branching trees with infinite depth, but only finitely many non-isomorphic subtrees.

Furthermore, since in this case trees "degenerate" to infinite regular lists, there exists an isomorphism between the structure of infinite regular trees and that of periodic[1] streams. This happens if the constructor on streams is interpreted in the obvious way: for all streams s, $n : s$ evaluates to the stream s' such that $s'(0) = n$ and $s'(i + 1) = s(i)$, for all natural numbers i.

For instance, the equation corresponding to the environment $x \mapsto 1 : 2 : x$ has as unique solution the infinite regular tree



which, in turn, corresponds to the unique solution s as stream, where $s(2i) = 1$, $s(2i + 1) = 2$, for all natural numbers i. In summary, both the existence of a unique solution and the equivalence of operational streams can be easily checked by relying on those of the corresponding regular trees.

Adding operators: expressive power. Allowing in environments other operators besides the constructor, the expressive power augments, since non-periodic streams can be represented by operational streams, that is, obtained as solutions of the equational systems, as we illustrate below.

For instance, if we consider also the stream operator of pointwise addition [+], then it is possible to define function `nat()` as follows:

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

The call `nat()` returns the stream which is the solution associated with x in the system of equations corresponding to $\{x \mapsto 0 : (x\,[+]\,y), y \mapsto 1 : y\}$.

In this case, the unique solution is the function s_x such that $s_x(i) = i$, for all natural numbers, that is, the stream of natural numbers, which is a non-periodic stream, and cannot be defined by equations which contain only the constructor. Hence, the expressive power has been increased.

A more involved example that can be expressed in our calculus is the non-periodic stream s such that $s(i) = i^n$, for all natural numbers i and a given exponent $n \geqslant 0$:

```
nat_to_pow(n) = if n <= 0 then repeat(1)
                          else nat_to_pow(n-1)[*]nat()
```

[1] A stream s is periodic iff there exist natural numbers $k \geqslant 0$, $p \geqslant 1$ such that $s(i + k) = s(i + k + p)$ for all natural numbers i.

where `[*]` is the pointwise multiplication.

For instance, the call `nat_to_pow(2)` returns the stream s_x such that $s_x(i) = i^2$, for all natural numbers i, which is the solution associated with x in the system of equations corresponding to $\{x \mapsto z\,[*]\,y\,[*]\,y, y \mapsto 0 : (y\,[+]\,z), z \mapsto 1 : z\}$.

Our calculus supports also an interleaving operator $\|$ for alternating the elements[2] of two streams. For instance, it is possible to define the Thue-Morse sequence [16] in a compact way:

```
thue_morse() = 0:one()
one() = 1:(one()‖zero())
zero() = 0:(zero()‖one())
```

The call `thue_morse()` returns the stream s_x which is the solution associated with x in the system of equations corresponding to $\{x \mapsto 0 : y, y \mapsto 1 : (y \parallel z), z \mapsto 0 : (z \parallel y)\}$. That is, for all natural numbers i, $s_x(i) = b$, where b is the parity bit of the binary encoding of i.

By using pointwise subtraction `[-]` and the postfix tail operator `^`, an alternative equivalent definition can be given:

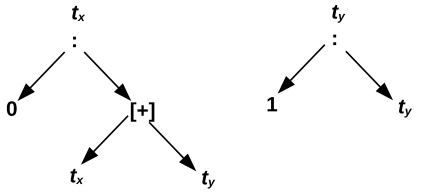
```
thue_morse2() =
  0:((repeat(1)[-]thue_morse2())‖(thue_morse2()^))
```

Adding operators: problems. Such an augmented expressive power[3] comes at a price: there is no longer an isomorphism between the structure of regular trees and that of streams. Hence, neither the existence of a unique solution nor the equivalence of operational streams can rely on those of the corresponding regular trees.

We first show that there are systems where equations are guarded (by the constructor or any added operator) that admit no unique solutions in the structure of streams.

Let us consider for instance $\{x \mapsto x[+]y, y \mapsto 1 : y\}$: there exists no stream solution to the equation associated with x. Conversely, $\{x \mapsto x[+]y, y \mapsto 0 : y\}$ admits an infinite set of streams which are solutions. However, since equations are guarded, in the structure of regular trees the equations above have unique solutions.

This first problem has been addressed in previous work [3,4,6], by enforcing further static checks on the equations which have been proved to be necessary and sufficient to ensure the unicity of solutions in the structure of streams. Under such further conditions, there is a unique solution both in the structure of trees and that of streams. For instance, considering again the environment $\{x \mapsto 0 : (x\,[+]\,y), y \mapsto 1 : y\}$, whose unique solution associated to x is the stream of natural numbers, the solutions as infinite regular trees t_x and t_y associated with x and y, respectively, can be depicted as follows:



However, it is easy to see that different trees can correspond to the same stream, see the example below.

The second problem, which is that specifically tackled in this work, again stems from the fact that the two structures of regular trees and streams are no longer isomorphic. As mentioned above, when only the constructor is considered, by virtue of the isomorphism, equality between periodic streams coincides with equality between the corresponding infinite regular trees.

Let us consider, for instance, the two different systems of equations corresponding to $\{x \mapsto 1 : x\}$ and $\{y \mapsto 1 : 1 : y\}$; they have as unique solution the same regular tree, and, hence, the same stream. This can be decided with an efficient sound and complete procedure corresponding, roughly, to the traversal of the two graphs representing the two regular trees, as happens, for instance, for cyclic terms in SWI-Prolog.

However, if one introduces other operators, as, for instance, the tail operator _^, then equality of regular trees becomes too strong, while the underlying equational theory for streams is non-trivial; for instance, the systems of equations corresponding to $\{x \mapsto 1 : x\}$ and $\{x \mapsto 1 : y, y \mapsto 1 : x^\wedge\}$ have the same unique solution associated with x as stream, but not as regular tree.

Since regular corecursion, and, hence, cycle detection, is based on equivalence between the operational streams returned by function calls to avoid non termination, if a suitable notion of equivalence is not adopted, then the increased expressive power which allows the representation of non-periodic streams is partially lost. Indeed, some function calls may not terminate since cycle detection fails.

Contribution and summary. In this paper, we provide an equivalence between operational streams which is coarser than equality of regular terms, and is *sound*, in the sense that equivalent operational streams define the same abstract stream; this is shown by proving that by accessing an arbitrary index of provably equivalent streams we get the same result. Moreover, this equivalence is complete when restricted to operational streams built by only the constructor and the tail operator (we shortly say *relatively complete*). In

[2] Formally, $s_1 \parallel s_2$ returns the stream s such that $s(2i) = s_1(i)$ and $s_2(2i+1) = s_2(i)$, for all natural numbers i.

[3] We refer to previous work [3,4,6] for a great variety of other interesting programming examples.

$\overline{fd}$	::=	$fd_1 \dots fd_n$	program
fd	::=	$f(\overline{x}) = se$	function declaration
e	::=	$se \mid ne \mid be$	expression
se	::=	$x \mid \texttt{if}\ be\ \texttt{then}\ se_1\ \texttt{else}\ se_2 \mid ne : se \mid se^{\wedge} \mid se_1\ op\ se_2 \mid f(\overline{e})$	stream expression
ne	::=	$x_n \mid se(ne) \mid ne_1\ op\ ne_2 \mid 0 \mid 1 \mid 2 \mid \dots$	numeric expression
be	::=	$x_b \mid \texttt{true} \mid \texttt{false} \mid \dots$	boolean expression
op	::=	$[nop] \mid \|$	binary stream operator
nop	::=	$+ \mid - \mid * \mid /$	arithmetic operation

Fig. 1. Stream calculus: syntax.

particular, when the two streams are built by only the constructor, checking the equivalence of two streams reduces to the procedure for the regular case.

The equivalence is expressed as a judgment defined by a coinductive inference system, where some rules are based on an auxiliary function for partial symbolic evaluation of the tail operator.

This description is quite natural, abstract, and convenient for the proofs of soundness and relative completeness, but non-algorithmic since partial symbolic evaluation of the tail operator might not terminate, and the coinductive interpretation of rules allows infinite derivation trees.

Therefore, to prove that the equivalence judgment is decidable, we show that the symbolic evaluation of the tail operator always terminates, and that derivation trees are always *regular*. The latter property allows us to use the standard technique, adopted in coinductive Logic Programming (co-LP) [26,27] and proved sound for an arbitrary inference system in [11], of transforming the coinductive inference system into an equivalent inductive one.

To sum up, we provide an equivalence which allows the identification of two calls in more cases than equality of trees, thus improving cycle detection, and prove that it is still sound and decidable. Completeness does not hold, meaning that there are operational streams which are not identified, though they denote the same abstract stream. The main reason is that the properties of arithmetic operators and interleaving are not taken into account, as we discuss in more detail in the Conclusion.

This paper is a largely extended version of [5], which did not include relative completeness and the decidability results mentioned above.

After reporting the calculus in Sect. 2, in Sect. 3 we provide the coinductive definition of equivalence and in Sect. 4 we prove its soundness and relative completeness. In Sect. 5 we prove the decidability results mentioned above. Finally, in Sect. 6 we summarize the contribution of the paper, and discuss related and further work. Appendix A reports the inductive definition of equivalence, and related examples.

2. Stream calculus

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

A program is a sequence of mutually recursive function declarations (see the productions for $\overline{fd}$ and fd in Fig. 1), for simplicity assumed to only return streams. Expressions are either stream, or numeric, or boolean expressions, with distinct sets of variables. 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, pointwise arithmetic operations and the interleaving operator ($\|$), giving a stream that is the strict interleaving of the elements of the arguments. Numeric expressions include $se(ne)$, with se stream expression and ne index expression,[4] denoting the access to an element of a stream. Dots denote other unspecified numeric and boolean expressions. We use $\overline{fd}$ to denote a sequence $fd_1, \dots, fd_n$ of function declarations, and analogously for other sequences.

The operational semantics is based on two key ideas:

1. streams are represented by means of 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 terms* s, built on top of stream variables, numeric values and the stream operators; consequently, a single stream is operationally represented by a pair (s, ρ), similarly as done with *capsules* [18] to support cyclic references. For instance, $(x, x \mapsto n : x)$ is the operational representation of the stream constantly equal to n.

We denote by $vars(\rho)$ the set of variables occurring in ρ, by $dom(\rho)$ its domain, by $fv(\rho)$ the set of its free variables, that is, $vars(\rho) \setminus dom(\rho)$, and say that ρ is *closed* if $fv(\rho) = \emptyset$, *open* otherwise, and analogously for a pair (s, ρ) representing a stream.

To obtain point (2), evaluation has an additional parameter which is a *call trace*, a map from function calls where arguments are values (*calls* for short in the following) into variables. Consider the following function declarations:

```
one_two() = 1:two_one()
two_one() = 2:one_two()
```

[4] 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.

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

Fig. 2. Stream calculus: values, call traces, and environments.

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

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

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

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

Fig. 3. Stream calculus: operational semantics (1).

and the call `one_two()`. The call trace is initially empty; when `one_two()` is invoked, it is added in the call trace with an associated fresh variable, say x. That is, the call trace becomes $\{\texttt{one\_two()} \mapsto x\}$. When evaluating the body of the function, another fresh variable y is generated for the intermediate call `two_one()`. The call trace becomes $\{\texttt{one\_two()} \mapsto x,\ \texttt{two\_one()} \mapsto y\}$. Thus, the recursive call `one_two()` can be detected as cyclic.

Values, call traces and environments are defined in Fig. 2. Note that there are two different notions of "value" for streams: an (open) stream term s plays the role of a value in the sense of an expression which does not need any evaluation, see rule (VAL) in Fig. 3, whereas an operational stream (s, ρ) is the final result of the evaluation of a stream expression.

The semantic judgment has the form $e, \rho, \tau \Downarrow (v, \rho')$, 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 (operational) stream obtained as result. The semantic judgements are also implicitly parametrized by a program, which is fixed and omitted.

The semantic judgment is defined in Fig. 3 and Fig. 4. In Fig. 3, 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. In rule (OP), $\rho \uplus \rho'$ denotes the union of two environments, which is well-defined if they have disjoint domains.

In Fig. 4, the rules for function call use a mechanism of cycle detection, similar to that in [2]. They are given in a modular way. That is, evaluation of arguments is handled by a separate rule (ARGS).

Rules for calls use the following auxiliary definitions:

- $\rho\{x \mapsto s\}$ is the environment which gives s on x, coincides with ρ elsewhere; we use an analogous notation for call traces.
- $se[\overline{v}/\overline{x}]$ is obtained by parallel substitution of variables $\overline{x}$ with values $\overline{v}$.
- $fbody(f)$ returns the pair of the parameters and the body of the declaration of f, if any, in the assumed program.

Moreover, rules for calls are parametric with respect to an *equivalence judgment* $v_1 \approx_\rho v_2$, stating that v_1 and v_2 are considered equivalent in the environment ρ. We expect this judgment to coincide with the identity for primitive boolean and numeric values, whereas different stream values could be identified, provided that they denote the same abstract stream (soundness of the equivalence). In Sect. 3 we will provide a sound definition of such judgment. The parametricity with respect to the equivalence consists, more in detail, in the fact that the notations $c \notin dom(\tau_{\approx\rho})$ and $c\ \tau_{\approx\rho}\ x$, used in the side condition of rule (INVK) and (COREC), respectively, rely on such judgment, as described below:

- two calls are equivalent if the function and the number of arguments are the same, and their arguments are respectively equivalent, formally:
  $c \approx_\rho c'$ iff $c = f(v_1, \ldots, v_n)$, $c' = f(v'_1, \ldots, v'_n)$ and $v_i \approx_\rho v'_i$ for all $i \in 1..n$
- look-up of a call in the call trace is performed *modulo equivalence*, formally the relation $\tau_{\approx\rho}$ is defined: $c\ \tau_{\approx\rho}\ x$ if $\tau(c') = x$, $c' \approx_\rho c$ for some c'. This means that rule (COREC) might lead to non-determinism since, in principle, different variables could be returned as result. However, the soundness result claimed in Theorem 4.3 ensures that this non-determinism does not affect the behavior of programs: function calls always return equivalent values even when (COREC) can return different variables.
- hence, $c \notin dom(\tau_{\approx\rho})$ if there is no $c' \in dom(\tau)$ such that $c \approx_\rho c'$.

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

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

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

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

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

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

Fig. 4. Stream calculus: operational semantics (2).

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, meaning 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 equivalence. At this point, the variable x is returned. However, there is no associated value in the environment yet; in other words, (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; for instance, x can be safely passed as an argument to a function call.

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

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 in the bottom part of the figure.

If the stream term is a variable, rule (AT-VAR), then the evaluation is propagated to the associated stream term 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 term; analogously, for odd indexes, the second rule is applied and the evaluation is propagated to the right-hand side stream term. The remaining rules are self-explanatory; for examples of derivations in this calculus, we point the reader to [3,4].

3. Equivalence of streams

In this section, we consider the problem of equivalence of (operational) streams. As discussed in the Introduction, and shown in the operational semantics in Fig. 4, this issue is relevant not only to provide an equivalence 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 stream $(x, x \mapsto 1 : x)$, then `incr_reg` is recursively called on $(x^\wedge, x \mapsto 1 : x)$ and cycle detection fails because the two results are not syntactically equal, although they denote the same abstract 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 obtained by considering equality in the theory of regular terms, as in [2]. This can be done by a coinductive definition[5] which computes the unfolding of variables by

[5] As further discussed in Sect. 5, for regular terms the coinductive definition can be turned into an equivalent inductive one.

$$
\begin{array}{lcll}
s & ::= & x \mid n : s \mid s^{\wedge} \mid s_1\ op\ s_2 & \text{(open) stream term} \\
op & ::= & [nop] \mid \| & \text{binary stream operator} \\
\rho & ::= & x_1 \mapsto s_1 \ldots x_n \mapsto s_n \quad (n \geqslant 0) & \text{environment}
\end{array}
$$

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

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

$$
\text{(OP)}\ \frac{s_1 \approx_\rho s_1' \quad s_2 \approx_\rho s_2'}{s_1\ op\ s_2 \approx_\rho s_1'\ op\ s_2'}
$$

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

$$
\text{(TAIL-TAIL)}\ \frac{\mathsf{Tail}_\rho(s) = s' \quad \mathsf{Tail}_\rho(s') = s''}{\mathsf{Tail}_\rho(s^{\wedge}) = s''}
$$

$$
\text{(TAIL-NOP)}\ \frac{}{\mathsf{Tail}_\rho(s_1[nop]s_2) = s_1^{\wedge}[nop]s_2^{\wedge}} \qquad \text{(TAIL-}\|\text{)}\ \frac{}{\mathsf{Tail}_\rho(s_1 \,\|\, s_2) = s_2 \,\|\, s_1^{\wedge}}
$$

Fig. 5. Equivalence check.

looking up their associated terms in the environment. This allows us to identify, e.g., streams returned by `ones()` and `altOnes()`, where `altOnes() = 1:1:altOnes()`. Indeed, in the environment of the form $\{x \mapsto 1 : x, y \mapsto 1 : 1 : y\}$ resulting from the evaluation, one can check by unfolding that the regular terms associated with x and y are the same. However, if one allows other operators in the equational systems, then equality of regular terms fails to identify, e.g., 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 equivalence 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 $(x, x \mapsto 1 : x)$.

On the other hand, our solution is admittedly incomplete to a large extent since there are equivalences which follow from the meaning of the arithmetic operators. For instance, given the environment $\rho = \{x \mapsto 1 : x, y \mapsto 2 : y, z \mapsto 3 : z\}$, we *do not derive* that $(x\,[+]\,y, \rho)$ is equivalent to (z, ρ).

The equivalence check is formalized by the judgment $s_1 \approx_\rho s_2$, which is defined in Fig. 5, where rules are interpreted *coinductively*, as denoted by the thick line, that is, derivable judgments are those which have an either finite or infinite proof tree [22].

In rules (VAR-L) and (VAR-R), a variable defined in the environment is equivalent to a stream term if the same holds for its associated stream term. In rule (VAR), a variable is equivalent to itself.

In rule (CONS), prepending the same element to equivalent streams gives equivalent streams.

In rule (OP), two streams defined using the same binary operator are equivalent if their arguments are respectively equivalent.

The most interesting rules are those handling the case when one of the two sides of the equivalence is of the form $s^{\wedge}$, which are the only ones applicable when the other side is not a variable. Indeed, in such case an attempt is made at computing (a stream term equivalent to) the tail of s, through the auxiliary function Tail_ρ defined in the bottom part of the figure.

Function Tail_ρ is inductively defined, that is, derivable judgments are those with a finite proof tree, and the base cases are: the constructor, rule (TAIL-CONS), where the result is simply the tail stream, the numeric binary operator, rule (TAIL-NOP), where the tail operator is applied to the two substreams, and the interleaving operator, rule (TAIL-$\|$), where the result is the interleaving of the second substream and the term obtained applying the tail operator to the first substream. In the other cases, the function is propagated as expected. In particular, in rule (TAIL-TAIL), to obtain the result we need two subsequent applications of Tail_ρ.

As mentioned above, the judgment $s_1 \approx_\rho s_2$ is defined by interpreting the rules in Fig. 5 coinductively, that is, infinite derivations are allowed. Such coinductive definition is the most natural and abstract for infinite objects such as streams, and is convenient for the proofs in Sect. 4, but is clearly non-algorithmic. It is possible to provide an alternative *inductive* definition by following the standard technique, adopted in co-LP [26,27] and proved sound for an arbitrary inference system in [11], of adding already considered judgments as *coinductive hypotheses*. Such an inductive definition, reported in the Appendix, is an important step towards an algorithm for equivalence (see Sect. 5).

Examples. We illustrate how the equivalence check works by some examples; here and in the following proof trees side conditions are written as premises to save space. The first one, in Fig. 6, shows the derivation of the equivalence $x \approx_\rho y^{\wedge\wedge}$ in the environment $\rho = \{x \mapsto 1 : x, y \mapsto 2 : 3 : 1 : y^{\wedge\wedge}\}$.

In this derivation, we applied rule (VAR-L) first, and then rule (TAIL-R) to get a stream value equivalent to $y^{\wedge\wedge}$; we could have applied the rules in the other order as well. The derivation of $\mathsf{Tail}_\rho(y^{\wedge}) = 1 : y^{\wedge\wedge}$ is shown in the bottom part of the figure. The result

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

$$\dfrac{\dfrac{\dfrac{}{\mathsf{Tail}_\rho(2 : 3 : 1 : y^{\wedge\wedge}) = 3 : 1 : y^{\wedge\wedge}}\,(\text{TAIL-CONS})}{\mathsf{Tail}_\rho(y) = 3 : 1 : y^{\wedge\wedge}}\,(\text{TAIL-VAR}) \qquad \dfrac{}{\mathsf{Tail}_\rho(3 : 1 : y^{\wedge\wedge}) = 1 : y^{\wedge\wedge}}\,(\text{TAIL-CONS})}{\mathsf{Tail}_\rho(y^\wedge) = 1 : y^{\wedge\wedge}}\,(\text{TAIL-TAIL})$$

Fig. 6. $x \approx_\rho y^{\wedge\wedge}$ with $\rho = \{x \mapsto 1 : x, y \mapsto 2 : 3 : 1 : y^{\wedge\wedge}\}$.

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

Fig. 7. $x \approx_\rho y$ with $\rho = \{x \mapsto 1 : (x\ [+]\ x), y \mapsto 1 : ((1 : y)\ [+]\ (1 : y))^\wedge\}$.

is computed by applying twice the tail operator, and unfolding y. Then, in the main derivation tree, rule (CONS) is applied since the first element of both streams is 1. After that, it is easy to see that there is a regular infinite derivation, denoted by the dots.

In the second example we show how the equivalence check deals with non-regular streams. Fig. 7 shows the derivation of $x \approx_\rho y$ with $\rho = \{x \mapsto 1 : (x\ [+]\ x), y \mapsto 1 : ((1 : y)\ [+]\ (1 : y))^\wedge\}$.

Here, both x and y denote the stream of all powers of 2. The derivation starts with the unfolding of variables x and y, followed by the application of rule (CONS). Then, the tail of the second component is computed, with the reduction tree shown as additional premise in rule (TAIL-R).

After that, the derivation continues by distributing the equivalence check to the substreams x and $(1 : y)^\wedge$. At this point, it is easy to see that there is a regular infinite derivation, after another application of rule (TAIL-R). For space reasons, we have omitted the derivation for the right premise of rule (OP) which is equal to the left one.

Examples of equivalences which cannot be derived with the rules in Fig. 5 are shown in Sect. 6, where the decidability status of the equivalence problem is discussed.

4. Soundness and (relative) completeness

We now prove the soundness of the equivalence check. That is, if we derive that two operational streams are equivalent, then they define the same abstract stream, that is, access to an arbitrary index will give the same result. Soundness of the equivalence check (Theorem 4.3) relies on soundness of the Tail judgment (Theorem 4.1).

Although we are mainly interested that the main theorems proved in this section hold for well-defined streams, some results are generalized to partially defined ones.

A stream (s, ρ) is well-defined if and only if for all $i \geqslant 0$, there exists n such that $at_\rho(s, i) = n$. We refer to related work [3,4,6] for a detailed study of the notion of well-defined stream.

In the proof of Theorem 4.1, we write "of the form tail or variable" to indicate streams of the form $s^\wedge$ or x, respectively. Moreover, we introduce the notation $at_\rho(s, i) \overset{k}{=} n$, meaning that the proof tree of $at_\rho(s, i) = n$ has depth k.

Theorem 4.1. *If $\mathsf{Tail}_\rho(s) = s'$ then, for all $i \geqslant 0$, $at_\rho(s^\wedge, i) \overset{k}{=} n$ implies $at_\rho(s', i) \overset{k'}{=} n$ with $k' \leqslant k$, and $k' < k$ if s' is of the form tail or variable.*

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

(TAIL-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 $i \geqslant 0$, $at_\rho(x^\wedge, i) \overset{k}{=} n$ implies $at_\rho(s, i) \overset{k'}{=} n$ with $k' \leqslant k$, and $k' < k$ if s is of the form tail or variable. We have that $at_\rho(x^\wedge, i) \overset{k}{=} n$ is necessarily derived by rule (AT-TAIL) from $at_\rho(x, i + 1) \overset{k-1}{=} n$ which, in turn, is necessarily derived by rule (AT-VAR) from $at_\rho(\rho(x), i + 1) \overset{k-2}{=} n$. Hence, by applying rule (AT-TAIL) $at_\rho(\rho(x)^\wedge, i) \overset{k-1}{=} n$ is derivable; therefore, by inductive hypothesis we have $at_\rho(s, i) \overset{k'}{=} n$ with $k' \leqslant k - 1$; since $k - 1 < k$ the thesis holds.

(TAIL-CONS) This is an axiom with conclusion $\mathsf{Tail}_\rho(n:s) = s$. We have to show that, for all $i \geqslant 0$, $at_\rho((n:s)^\wedge, i) \overset{k}{=} n$ implies $at_\rho(s, i) \overset{k'}{=} n$ with $k' \leqslant k$ and $k' < k$ if s is of the form tail or variable. We have that $at_\rho((n:s)^\wedge, i) \overset{k}{=} n$ is necessarily derived by rule (AT-TAIL) from $at_\rho(n:s, i+1) \overset{k-1}{=} n$ which, in turn, is necessarily derived by rule (AT-CONS-SUCC) from $at_\rho(s, i) \overset{k-2}{=} n$; since $k-2 < k$ the thesis holds.

(TAIL-TAIL) $\mathsf{Tail}_\rho(s) = s', \mathsf{Tail}_\rho(s') = s''$ are the premises, and $\mathsf{Tail}_\rho(s^\wedge) = s''$ the conclusion. We have to show that, for all $i \geqslant 0$, $at_\rho(s^{\wedge\wedge}, i) \overset{k}{=} n$ implies $at_\rho(s'', i) \overset{k''}{=} n$ with $k'' \leqslant k$ and $k'' < k$ if s'' is of the form tail or variable. We have that $at_\rho(s^{\wedge\wedge}, i) \overset{k}{=} n$ is necessarily derived by rule (AT-TAIL) from $at_\rho(s^\wedge, i+1) \overset{k-1}{=} n$; therefore, by inductive hypothesis on the first premise we have $at_\rho(s', i+1) \overset{k'}{=} n$, with $k' \leqslant k-1$. Hence, by applying rule (AT-TAIL) $at_\rho(s'^\wedge, i) \overset{k'+1}{=} n$ is derivable with $k'+1 \leqslant k$. By inductive hypothesis on the second premise we have $at_\rho(s'', i) \overset{k''}{=} n$, with $k'' \leqslant k'+1 \leqslant k$; furthermore, if s'' is of the form tail or variable, then, again by inductive hypothesis on the second premise, $k'' < k'+1 \leqslant k$, hence $k'' < k$.

(TAIL-NOP) This is an axiom with conclusion $\mathsf{Tail}_\rho(s_1[nop]s_2) = s_1^\wedge[nop]s_2^\wedge$. Since in this case $s_1^\wedge[nop]s_2^\wedge$ is not of the form tail or variable, we only have to show that, for all $i \geqslant 0$, $at_\rho((s_1[nop]s_2)^\wedge, i) \overset{k}{=} n$ implies $at_\rho(s_1^\wedge[nop]s_2^\wedge, i) \overset{k'}{=} n$ with $k' \leqslant k$. We have that $at_\rho((s_1[nop]s_2)^\wedge, i) \overset{k}{=} n$ is necessarily derived by rule (AT-TAIL) from $at_\rho(s_1[nop]s_2, i+1) \overset{k-1}{=} n$ which, in turn, is necessarily derived by rule (AT-NOP) from $at_\rho(s_1, i+1) \overset{k_1}{=} n_1$ and $at_\rho(s_2, i+1) \overset{k_2}{=} n_2$, with $n_1\ nop\ n_2 = n$, $\max(k_1, k_2) = k-2$. Hence, by applying rule (AT-TAIL) $at_\rho(s_1^\wedge, i) \overset{k_1+1}{=} n_1$ and $at_\rho(s_2^\wedge, i) \overset{k_2+1}{=} n_2$ are derivable. Hence, by applying rule (AT-NOP) $at_\rho(s_1^\wedge[nop]s_2^\wedge, i) \overset{k'}{=} n$ is derivable, with $k' = \max(k_1+1, k_2+1)+1 = \max(k_1, k_2)+2 = k$.

(TAIL-‖) This is an axiom with conclusion $\mathsf{Tail}_\rho(s_1 \parallel s_2) = s_2 \parallel s_1^\wedge$. Since in this case $s_2 \parallel s_1^\wedge$ is not of the form tail or variable, we only have to show that, for all $i \geqslant 0$, $at_\rho((s_1 \parallel s_2)^\wedge, i) \overset{k}{=} n$ implies $at_\rho(s_2 \parallel s_1^\wedge, i) \overset{k'}{=} n$ with $k' \leqslant k$. We have that $at_\rho((s_1 \parallel s_2)^\wedge, i) \overset{k}{=} n$ is necessarily derived by rule (AT-TAIL) from $at_\rho(s_1 \parallel s_2, i+1) \overset{k-1}{=} n$; the proof is by case analysis on the parity of i:

i **even:** $at_\rho(s_1 \parallel s_2, i+1) \overset{k-1}{=} n$ is necessarily derived by rule (AT-‖-ODD) from $at_\rho(s_2, \frac{i}{2}) \overset{k-2}{=} n$. Hence, by applying rule (AT-‖-EVEN) $at_\rho(s_2 \parallel s_1^\wedge, i) \overset{k'}{=} n$ is derivable, with $k' = k-1 < k$.

i **odd:** $at_\rho(s_1 \parallel s_2, i+1) \overset{k-1}{=} n$ is necessarily derived by rule (AT-‖-EVEN) from $at_\rho(s_1, \frac{i+1}{2}) \overset{k-2}{=} n$. Hence, by applying rule (AT-TAIL) $at_\rho(s_1^\wedge, \frac{i-1}{2}) \overset{k-1}{=} n$ and by applying rule (AT-‖-ODD) $at_\rho(s_2 \parallel s_1^\wedge, i) \overset{k'}{=} n$ is derivable, with $k' = k$. □

In the following, given the pairs of natural numbers (k_1, k_2) and (k_1', k_2'), we denote with $(k_1, k_2) \leqslant (k_1', k_2')$ the componentwise order defined by $(k_1, k_2) \leqslant (k_1', k_2')$ iff $k_1 \leqslant k_1'$ and $k_2 \leqslant k_2'$, and analogously for the strict variant $<$. The following lemma, which relies on Theorem 4.1, is instrumental to the proof of soundness.

Lemma 4.2. *Let d be a derivation of height > 0 for $s_1 \approx_\rho s_2$; if $at_\rho(s_1, i) \overset{k_1}{=} n_1$, and $at_\rho(s_2, i) \overset{k_2}{=} n_2$ hold with $(k_1, k_2) > (0, 0)$, then one of the following holds:*

1. *either in d there exists a judgment $s_1' \approx_\rho s_2'$ such that,*
   for some i', k_1', k_2',

   $$at_\rho(s_1', i') \overset{k_1'}{=} n_1,$$
   $$at_\rho(s_2', i') \overset{k_2'}{=} n_2,$$
   $$(k_1', k_2') < (k_1, k_2)$$

2. *or in d there exist two judgments $s_1' \approx_\rho s_2'$ and $s_1'' \approx_\rho s_2''$ such that,*
   for some nop, k_1', k_2', k_1'', k_2'', n_1', n_2', n_1'', n_2'',

   $$n_1 = n_1'\ nop\ n_1'',\ n_2 = n_2'\ nop\ n_2'',$$
   $$at_\rho(s_1', i) \overset{k_1'}{=} n_1',\ at_\rho(s_1'', i) \overset{k_1''}{=} n_1'',$$
   $$at_\rho(s_2', i) \overset{k_2'}{=} n_2',\ at_\rho(s_2'', i) \overset{k_2''}{=} n_2'',$$
   $$(k_1', k_2'), (k_1'', k_2'') < (k_1, k_2).$$

Proof. By case analysis on the rule applied for the root.

case (VAR-L): in this case $s_1 = x$ for some x, hence we have $x \approx_\rho s_2$ and property 1 holds for the premise $\rho(x) \approx_\rho s_2$. Indeed, we have $at_\rho(x,i) \overset{k_1}{=} n_1$, $at_\rho(s_2,i) \overset{k_2}{=} n_2$ and, since the former judgment has been necessarily derived by rule (AT-VAR), $at_\rho(\rho(x),i) \overset{k_1-1}{=} n_1$. Hence the claim holds because $(k_1-1,k_2) < (k_1,k_2)$.
case (VAR-R): symmetrical to the previous case.
case (VAR): this case is vacuous because by hypothesis the derivation d must have height > 0.
case (CONS): in this case $s_1 = m : s_1'$ and $s_2 = m : s_2'$ for some m, s_1', s_2', hence we have $m : s_1' \approx_\rho m : s_2'$ and property 1 holds for the premise $s_1' \approx_\rho s_2'$. Indeed, we have $at_\rho(m : s_1', i) \overset{k_1}{=} n_1$, $at_\rho(m : s_2', i) \overset{k_2}{=} n_2$ and, since $(k_1,k_2) > (0,0)$, we have that $i > 0$ and that these judgments have been necessarily derived by rule (AT-CONS-SUCC); therefore, $at_\rho(s_1', i-1) \overset{k_1-1}{=} n_1$, $at_\rho(s_2', i-1) \overset{k_2-1}{=} n_2$. Hence the claim holds because $(k_1-1, k_2-1) < (k_1,k_2)$.
case (OP): the proof proceeds by cases on the binary operator *op*.
[*nop*]: in this case for some *nop*, s_1', s_1'', s_2', s_2'' $s_1 = s_1'[nop]\ s_1''$ and $s_2 = s_2'[nop]\ s_2''$, hence we have $s_1'\ [nop]\ s_1'' \approx_\rho s_2'\ [nop]\ s_2''$ and property 2 holds for the premises $s_1' \approx_\rho s_2'$ and $s_1'' \approx_\rho s_2''$. Indeed, we have $at_\rho(s_1'\ [nop]\ s_1'', i) \overset{k_1}{=} n_1$, $at_\rho(s_2'\ [nop]\ s_2'', i) \overset{k_2}{=} n_2$ and, since these judgments have been necessarily derived by rule (AT-NOP), $at_\rho(s_1', i) \overset{k_1'}{=} n_1'$, $at_\rho(s_1'', i) \overset{k_1''}{=} n_1''$, $at_\rho(s_2', i) \overset{k_2'}{=} n_2'$, $at_\rho(s_2'', i) \overset{k_2''}{=} n_2''$, $n_1 = n_1'\ nop\ n_1''$, $n_2 = n_2'\ nop\ n_2''$, $k_1', k_1'' < k_1$, $k_2', k_2'' < k_2$. Hence the claim holds because $(k_1', k_2'), (k_1'', k_2'') < (k_1,k_2)$.
$\|$: in this case $s_1 = s_1' \parallel s_1''$ and $s_2 = s_2' \parallel s_2''$ for some s_1', s_1'', s_2', s_2'', hence we have $s_1' \parallel s_1'' \approx_\rho s_2' \parallel s_2''$. If i is even, then property 1 holds for the premise $s_1' \approx_\rho s_2'$. Indeed, we have $at_\rho(s_1' \parallel s_1'', i) \overset{k_1}{=} n_1$, $at_\rho(s_2' \parallel s_2'', i) \overset{k_2}{=} n_2$ and, since these judgments have been necessarily derived by rule (AT-$\|$-EVEN), $at_\rho(s_1', \frac{i}{2}) \overset{k_1-1}{=} n_1$, $at_\rho(s_2', \frac{i}{2}) \overset{k_2-1}{=} n_2$. Hence the claim holds because $(k_1-1, k_2-1) < (k_1,k_2)$.
If i is odd, then in a similar way one can prove that property 1 holds for the premise $s_1'' \approx_\rho s_2''$.
Remark: the proof for the previous cases (VAR-L), (VAR-R), (CONS), and (OP) shows, independently of the length of d, that the claim is always satisfied by a judgment at depth 1 in d.
case (TAIL-L): in this case $s_1 = s_1'^\wedge$ for some s_1', hence we have $s_1'^\wedge \approx_\rho s_2$, the side condition $\mathsf{Tail}_\rho(s_1') = s'$ and the premise $s' \approx_\rho s_2$. Moreover $at_\rho(s_1'^\wedge, i) \overset{k_1}{=} n_1$ and $at_\rho(s_2, i) \overset{k_2}{=} n_2$. By Theorem 4.1 $at_\rho(s', i) \overset{k_1'}{=} n_1$ with $k_1' \leqslant k_1$; furthermore, if s' is of the form variable or tail, then property 1 holds for $s' \approx_\rho s_2$ since $k_1' < k_1$, hence $(k_1', k_2) < (k_1,k_2)$. Hence, as happens for the previous cases, the claim always holds for a judgment at depth 1 in d.
If s' is not of the form variable or tail, then we prove that there exists a judgment at depth 2 or 3 in d satisfying property 1 or 2; that is, such a judgment is at depth 1 or 2 in the derivation of $s' \approx_\rho s_2$. For such a judgment the only applicable rules are (VAR-R), (CONS), (OP), or (TAIL-R). For all these rules, except for (TAIL-R), the proofs of the cases above already show that there exists a judgment at depth 1 in the derivation of $s' \approx_\rho s_2$ satisfying property 1 or 2, hence the claim holds for the derivation d of $s_1'^\wedge \approx_\rho s_2$ for a judgment at depth 2, because $at_\rho(s', i) \overset{k_1'}{=} n_1$ holds with $k_1' \leqslant k_1$ by Theorem 4.1.
If the rule applied for $s' \approx_\rho s_2$ is (TAIL-R), then $s_2 = s_2'^\wedge$ for some s_2'. We have the side condition $\mathsf{Tail}_\rho(s_2') = s''$ and the premise $s' \approx_\rho s''$; by reasoning similarly as done for (TAIL-L), by Theorem 4.1 $at_\rho(s'', i) \overset{k_2'}{=} n_2$ with $k_2' \leqslant k_2$; furthermore, if s'' is of the form variable or tail, then property 1 holds for $s' \approx_\rho s''$ because $k_2' < k_2$, $at_\rho(s', i) \overset{k_1'}{=} n_1$ holds with $k_1' \leqslant k_1$ by Theorem 4.1, hence $(k_1', k_2') < (k_1,k_2)$ and the claim holds for a judgment at depth 2 in d.
If s'' is not of the form variable or tail, then we prove that there exists a judgment at depth 3 in d satisfying property 1 or 2; that is, such a judgment is at depth 1 in the derivation of $s' \approx_\rho s''$. For such a judgment the only applicable rules are (CONS) or (OP); again, for these rules the proofs for the cases above already show that there exists a judgment at depth 1 in the derivation of $s' \approx_\rho s''$ satisfying property 1 or 2, hence the claim holds for the derivation d of $s_1'^\wedge \approx_\rho s_2$ for a judgment at depth 3, because $at_\rho(s', i) \overset{k_1'}{=} n_1$, $at_\rho(s'', i) \overset{k_2'}{=} n_2$ hold with $k_1' \leqslant k_1$ and $k_2' \leqslant k_2$ by Theorem 4.1.
case (TAIL-R): symmetrical to the case (TAIL-L). □

Even though it is not needed for the proof of soundness that follows, for sake of clarity it is worth noting some details of the proof of Lemma 4.2:

- the claim always holds for judgments at depth $\leqslant 3$ in the derivation d of $s_1 \approx_\rho s_2$;
- the proof is not by induction and is independent from the height of d, since it inspects the top of d limited to depth 3;
- property 2 is verified when $s_1 = s_1'\ [nop']\ s_1''$ and $s_2 = s_2'\ [nop']\ s_2''$, where the witness for the existentially quantified variable *nop* is nop'.

Theorem 4.3 *(Soundness of equivalence). For all $i \geqslant 0$, if $at_\rho(s_1,i) = n_1$, $at_\rho(s_2,i) = n_2$, and $s_1 \approx_\rho s_2$, then $n_1 = n_2$.*

Proof. Assume that $at_\rho(s_1,i) \overset{k_1}{=} n_1$, $at_\rho(s_2,i) \overset{k_2}{=} n_2$. We first consider the simple case where the derivation for $s_1 \approx_\rho s_2$ has height 0, that is, rule (VAR) is applied, and hence $s_1 = s_2 = x$ for some x; in this case we conclude by the fact that, by definition, the judgment $at_\rho(s,i) = n$ is deterministic, therefore $at_\rho(x,i) = n_1$ and $at_\rho(x,i) = n_2$ implies $n_1 = n_2$.

If the derivation for $s_1 \approx_\rho s_2$ has height > 0, then the proof proceeds by induction on the well-founded componentwise order on the pairs (k_1,k_2).

Base We consider the pair $(0,0)$, that is, the case when the two judgments are derived by the unique axiom (AT-CONS-0), hence, $i = 0$, $s_1 = n_1 : s_1'$, $s_2 = n_2 : s_2'$ for some s_1', s_2'. Moreover, we have $n_1 : s_1' \approx_\rho n_2 : s_2'$, which has been necessarily derived by rule (CONS), hence $n_1 = n_2$.

Inductive step We consider pairs (k_1,k_2) such that $(k_1,k_2) > (0,0)$, hence Lemma 4.2 can be applied.

If property 1 of the lemma holds, then we have that in the derivation of $s_1 \approx_\rho s_2$ there exists a judgment $s_1' \approx_\rho s_2'$ such that $at_\rho(s_1',i') \overset{k_1'}{=} n_1$, $at_\rho(s_2',i') \overset{k_2'}{=} n_2$, $(k_1',k_2') < (k_1,k_2)$ for some i', k_1', k_2', hence we can conclude by inductive hypothesis that $n_1 = n_2$.

If property 2 of the lemma holds, then we have that in the derivation of $s_1 \approx_\rho s_2$ there exist two judgments $s_1' \approx_\rho s_2'$ and $s_1'' \approx_\rho s_2''$ such that $at_\rho(s_1',i) \overset{k_1'}{=} n_1'$, $at_\rho(s_1'',i) \overset{k_1''}{=} n_1''$, $at_\rho(s_2',i) \overset{k_2'}{=} n_2'$, $at_\rho(s_2'',i) \overset{k_2''}{=} n_2''$, $n_1 = n_1' \ nop \ n_1''$, $n_2 = n_2' \ nop \ n_2''$, $(k_1',k_2'),(k_1'',k_2'') < (k_1,k_2)$ for some nop, k_1', k_2', k_1'', k_2'', n_1', n_2', n_1'', n_2''. By inductive hypothesis we deduce that $n_1' = n_2'$ and $n_1'' = n_2''$, hence $n_1 = n_1' \ nop \ n_1'' = n_2' \ nop \ n_2'' = n_2$. □

Although we are mainly interested that Theorem 4.3 holds for well-defined streams, its claim is rather general, because it covers also the case where the streams (s_0,ρ) and (s_0',ρ) are only partially well-defined, that is, $at_\rho(s_0,i)$ and $at_\rho(s_0',i)$ are defined only for some index i, but not for all indexes; consider, for instance, $0 : x$ and $0 : y$ in the environment $\rho = x \mapsto x, y \mapsto y$. Note also that, if $\rho = x \mapsto x, y \mapsto 0 : y$, then $x \approx_\rho y$ is derivable although (x,ρ) is undefined for any index and (y,ρ) is well-defined; this does not break the theorem, since in this case the hypotheses are not verified.

The following theorem proves a partial result of completeness by restricting the calculus to streams built by only using variables, and the constructor and tail operators.

The proof of the theorem is based on the technique that we quickly recall: let C be a claim of the form "for all x, $P(x)$ implies $Q(x)$", where P is a predicate and Q is a judgment defined coinductively. Then C can be proved by *coinduction* on the definition of Q in the following way: for all x satisfying P, one has to show that there always exists a rule r of Q such that x is the conclusion of r and all corresponding premises of r satisfy P as well. This is a consequence of the fact that the set of elements satisfying Q, that is, derivable with a possibly infinite proof tree, coincides with the greatest fixed-point of the inference operator F_Q derived from the rules defining Q:

$F(X) = \{y \mid \exists$ rule r of Q such that premises$(r) \in X$ and $y \in$conclusion$(r)\}$.

The interested reader can refer to the related literature for further technical details [23,15,22].

Theorem 4.4 *(Relative completeness of equivalence). Let s_0, s_0' and ρ be such that they only contain variables, and the constructor and tail operators. If, for all $i \geqslant 0$, $at_\rho(s_0,i) = at_\rho(s_0',i)$, then $s_0 \approx_\rho s_0'$.*

Proof. The proof is by coinduction on the rules defining $s_0 \approx_\rho s_0'$. Let us define predicate P such that $P(s_0,s_0')$ holds iff, for all $i \geqslant 0$, $at_\rho(s_0,i) = at_\rho(s_0',i)$ and let Q coincide with the judgment $\approx_\rho$; then the claim of the theorem has the form introduced above "for all x, $P(x)$ implies $Q(x)$", where Q is defined coinductively, and we have to show that, for each (s_0,s_0') such that $P(s_0,s_0')$ holds there is at least an applicable rule with consequence $s_0 \approx_\rho s_0'$ such that for every corresponding premise $s_1 \approx_\rho s_1'$ $P(s_1,s_1')$ holds as well. This is shown by case analysis on s_0 and s_0'.

Case $s_0 = x$ Rule (VAR-L) has a consequence of this the form with premise $\rho(x) \approx_\rho s_0'$; we show that $P(\rho(x),s_0')$ holds. Indeed, by hypothesis, for all $i \geqslant 0$ $at_\rho(x,i) = at_\rho(s_0',i)$ and necessarily (AT-VAR) has been used, therefore, for all $i \geqslant 0$ $at_\rho(\rho(x),i) = at_\rho(s_0',i)$.

Case $s_0' = x$ Symmetric to the previous one.

Case $s_0 = s^\wedge{}_1$ Rule (TAIL-L) has a consequence of this form with premise $s' \approx_\rho s_0'$ but can only be applied if the side condition $\mathsf{Tail}_\rho(s_1) = s'$ holds for some s'. By hypothesis, for all $i \geqslant 0$ $at_\rho(s_1^\wedge,i) = at_\rho(s_0',i)$; necessarily (AT-TAIL) has been used, therefore, for all $i \geqslant 0$ $at_\rho(s_1,i+1) = at_\rho(s_0',i)$, therefore by Theorem 5.1 there exists s' such that $\mathsf{Tail}_\rho(s_1) = s'$, therefore rule (TAIL-L) is applicable. By Theorem 4.1 for all $i \geqslant 0$, $at_\rho(s_1^\wedge,i) = n$ implies $at_\rho(s',i) = n$, hence, for all $i \geqslant 0$ $at_\rho(s',i) = at_\rho(s_0',i)$, hence $P(s',s_0')$ holds.

Case $s_0' = s_1'^\wedge$ Symmetric to the previous one.

Case $s_0 = n : s_1$, $s_0' = n' : s_1'$ Rule (CONS) has a consequence of this form with premise $s_1 \approx_\rho s_1'$ but can only be applied when $n = n'$. By hypothesis, for all $i \geqslant 0$ $at_\rho(n : s_1,i) = at_\rho(n' : s_1',i)$; necessarily rule (AT-CONS-0) has been used when $i = 0$, therefore $n = n'$ and rule (CONS) is applicable; when $i > 0$ rule (AT-CONS-SUCC) has been necessarily used, therefore for all $i \geqslant 0$ $at_\rho(s_1,i) = at_\rho(s_1',i)$, hence $P(s_1,s_1')$ holds. □

5. Towards an algorithm for equivalence

In this section our aim is to show that, for well-defined streams, derivations for the coinductive inference system in Fig. 5 are always regular and the tail of an operational stream can always be effectively computed symbolically. This is an important step to show that an algorithm for equivalence can be driven by the rules in Fig. 5. To do so, we need to address two separate issues, that is, to prove that

1. function Tail always terminates
2. all derivation trees for $\approx_\rho$ are regular, that is, they consist of a finite set of judgments.

The following theorem proves claim (1) under a minimal hypothesis of well-definedness, that is, that access to at least one index is successful.

Theorem 5.1. *If, for some $i \geqslant 0$, $at_\rho(s_0, i) = n$, then there exists s' such that $\mathsf{Tail}_\rho(s_0) = s'$.*

Proof. Assume that $at_\rho(s_0, i) \overset{k}{=} n$. The proof is by induction on k.

Base If $k = 0$, then $s_0 = n : s$ and $i = 0$; in this case the thesis trivially holds because the claim can be obtained from the corresponding axiom (TAIL-CONS) which is always applicable.

Inductive step Case analysis on the form of s_0.

- **Case x** By hypothesis, we have that $at_\rho(x, i) \overset{k}{=} n$; by rule (AT-VAR) $at_\rho(\rho(x), i) \overset{k-1}{=} n$ is necessarily derived and, by inductive hypothesis, there exists s' such that $\mathsf{Tail}_\rho(\rho(x)) = s'$, and from this we have the thesis by applying rule (TAIL-VAR).
- **Cases $n : s$, s_1 *nop* s_2, $s_1 \| s_2$** In these cases the thesis trivially holds because the claim can be obtained from the corresponding axioms (TAIL-CONS), (TAIL-NOP) and (TAIL-$\|$), which are always applicable.
- **Case $s^\wedge$** By hypothesis, we have that $at_\rho(s^\wedge, i) \overset{k}{=} n$; by rule (AT-TAIL) $at_\rho(s, i+1) \overset{k-1}{=} n$ is necessarily derived and, by inductive hypothesis, there exists s' such that $\mathsf{Tail}_\rho(s) = s'$ and $at_\rho(s', i) \overset{k'}{=} n$ by Theorem 4.1.
  If $k' < k$, then by inductive hypothesis there exists s'' such that $\mathsf{Tail}_\rho(s') = s''$ and we can conclude the thesis by applying rule (TAIL). Otherwise, by Theorem 4.1 s' is **not** of the form tail or variable, hence, as already shown above, there exists s'' such that $\mathsf{Tail}_\rho(s') = s''$ since the corresponding axiom can be always applied, therefore also when $k' \not< k$ the thesis follows by applying rule (TAIL). □

To prove claim (2) means to show that any derivation for a judgment $s \approx_\rho s'$ involves only a finite set of pairs. To this end, we need some auxiliary definitions and lemmas.

First of all, we define the two functions $\mathsf{max\_tails}_\rho$ and $\mathsf{bin\_ops}_\rho$, parameterized on an environment ρ, which take an open stream term s as argument, and return a natural number.

Both functions are defined by structural recursion, meaning that, on a compound term, they are recursively called on subterms; however, there is no base case ensuring termination, since, when a variable is found, the functions are recursively called on the associated term in the environment. Hence, to ensure termination, the functions are defined in terms of two auxiliary functions with an additional parameter which is used for accumulating into a set V the already visited variables, similarly as it happens in a graph traversal.[6] Initially such a set is empty, therefore, the two functions $\mathsf{max\_tails}_\rho$ and $\mathsf{bin\_ops}_\rho$ are simply defined in terms of the auxiliary functions given in Fig. 8 as follows: $\mathsf{max\_tails}_\rho(s) = \mathsf{max\_tails}_\rho(s, \emptyset)$ and $\mathsf{bin\_ops}_\rho(s) = \mathsf{bin\_ops}_\rho(s, \emptyset)$.

According to the definitions in Fig. 8, $\mathsf{max\_tails}_\rho(s)$ returns the maximal number of nested tail operators found during the traversal, and $\mathsf{bin\_ops}_\rho(s)$ the total number of binary operators (that is, stream constructor, pointwise arithmetic operators, and interleaving) found during the traversal. The analogy with a graph traversal could be made precise by defining the graph associated with (s, ρ) as that where nodes are operators, and there is an edge from an operator to all its subterms, and from a variable to its associated term in ρ. In this view, $\mathsf{max\_tails}_\rho(s)$ is the maximal number of (nodes which are) tail operators in a path from s to the second occurrence of a variable, and $\mathsf{bin\_ops}_\rho(s)$ is the total number of binary operators in all paths from s to the second occurrence of a variable..

The following lemmas are about a generalized version of $\mathsf{Tail}_\rho(s_0) = s'$, of shape $\mathsf{V} \vdash \mathsf{Tail}_\rho(s_0) = s'$, defined in Fig. 9. The set V of variables plays the same role as in the auxiliary functions above.

Lemma 5.2. *If $\mathsf{V} \vdash \mathsf{Tail}_\rho(s_0) = s'$ and $\mathsf{max\_tails}_\rho(s_0, \mathsf{V}) = k$, then we have $\mathsf{max\_tails}_\rho(s', \mathsf{V}) = k'$ and $k' \leqslant k + 1$.*

Proof. By induction on the rules defining $\mathsf{V} \vdash \mathsf{Tail}_\rho(s_0) = s'$.

(V-TAILS-CONS) By hypothesis, $\mathsf{V} \vdash \mathsf{Tail}_\rho(n : s) = s$ and $\mathsf{max\_tails}_\rho(n : s, \mathsf{V}) = k$. To derive this latter hypothesis we must have applied rule (CONS-MAXTAILS), so we also know that $\mathsf{max\_tails}_\rho(s, \mathsf{V}) = k$. The thesis immediately follows from this fact.

[6] This is also analogous to keeping the call trace in the operational semantics in Sect. 2.

$$\text{(VAR-MAXTAILS)}\quad \mathsf{max\_tails}_\rho(x,\mathsf{V}) = \begin{cases} 0 & \text{if } x \in \mathsf{V} \text{ or } x \notin dom(\rho) \\ \mathsf{max\_tails}_\rho(\rho(x), \mathsf{V} \cup \{x\}) & \text{otherwise} \end{cases}$$

$$\text{(CONS-MAXTAILS)}\quad \mathsf{max\_tails}_\rho(n:s,\mathsf{V}) = \mathsf{max\_tails}_\rho(s,\mathsf{V})$$

$$\text{(TAIL-MAXTAILS)}\quad \mathsf{max\_tails}_\rho(s^\wedge,\mathsf{V}) = \mathsf{max\_tails}_\rho(s,\mathsf{V}) + 1$$

$$\text{(OP-MAXTAILS)}\quad \mathsf{max\_tails}_\rho(s_1\ op\ s_2,\mathsf{V}) = \max(\mathsf{max\_tails}_\rho(s_1,\mathsf{V}), \mathsf{max\_tails}_\rho(s_2,\mathsf{V}))$$

$$\text{(VAR-BINOPS)}\quad \mathsf{bin\_ops}_\rho(x,\mathsf{V}) = \begin{cases} 0 & \text{if } x \in \mathsf{V} \text{ or } x \notin dom(\rho) \\ \mathsf{bin\_ops}_\rho(\rho(x), \mathsf{V} \cup \{x\}) & \text{otherwise} \end{cases}$$

$$\text{(CONS-BINOPS)}\quad \mathsf{bin\_ops}_\rho(n:s,\mathsf{V}) = \mathsf{bin\_ops}_\rho(s,\mathsf{V}) + 1$$

$$\text{(TAIL-BINOPS)}\quad \mathsf{bin\_ops}_\rho(s^\wedge,\mathsf{V}) = \mathsf{bin\_ops}_\rho(s,\mathsf{V})$$

$$\text{(OP-BINOPS)}\quad \mathsf{bin\_ops}_\rho(s_1\ op\ s_2,\mathsf{V}) = \mathsf{bin\_ops}_\rho(s_1,\mathsf{V}) + \mathsf{bin\_ops}_\rho(s_2,\mathsf{V}) + 1$$

Fig. 8. Auxiliary functions.

$$\text{(V-VAR-AX)}\ \frac{}{\mathsf{V} \vdash \mathsf{Tail}_\rho(x) = x}\ x \in \mathsf{V} \qquad \text{(V-VAR)}\ \frac{\mathsf{V} \cup \{x\} \vdash \mathsf{Tail}_\rho(\rho(x)) = s}{\mathsf{V} \vdash \mathsf{Tail}_\rho(x) = s}\ x \notin \mathsf{V}$$

$$\text{(V-TAIL-CONS)}\ \frac{}{\mathsf{V} \vdash \mathsf{Tail}_\rho(n:s) = s} \qquad \text{(V-TAIL)}\ \frac{\mathsf{V} \vdash \mathsf{Tail}_\rho(s) = s' \quad \mathsf{V} \vdash \mathsf{Tail}_\rho(s') = s''}{\mathsf{V} \vdash \mathsf{Tail}_\rho(s^\wedge) = s''}$$

$$\text{(V-NOP)}\ \frac{}{\mathsf{V} \vdash \mathsf{Tail}_\rho(s_1[nop]s_2) = s_1^\wedge[nop]s_2^\wedge} \qquad \text{(V-}\|\text{)}\ \frac{}{\mathsf{V} \vdash \mathsf{Tail}_\rho(s_1 \parallel s_2) = s_2 \parallel s_1^\wedge}$$

Fig. 9. Generalized definition of Tail.

(V-TAIL) By hypothesis, $\mathsf{V} \vdash \mathsf{Tail}_\rho(s^\wedge) = s''$ and $\mathsf{max\_tails}_\rho(s^\wedge, \mathsf{V}) = k$. To derive this latter hypothesis we must have applied rule (TAIL-MAXTAILS), so we also know that $\mathsf{max\_tails}_\rho(s, \mathsf{V}) = k - 1$; furthermore, by rule (V-TAIL), $\mathsf{V} \vdash \mathsf{Tail}_\rho(s) = s'$, therefore by inductive hypothesis, we know that $\mathsf{max\_tails}_\rho(s', \mathsf{V}) = k'$ with $k' \leqslant k$. Again, by rule (V-TAIL), $\mathsf{V} \vdash \mathsf{Tail}_\rho(s') = s''$, therefore by inductive hypothesis $\mathsf{max\_tails}_\rho(s'', \mathsf{V}) = k''$ with $k'' \leqslant k' + 1 \leqslant k + 1$, hence we get the thesis.

(V-VAR-AX) By hypothesis, we have $\mathsf{V} \vdash \mathsf{Tail}_\rho(x) = x$ with $x \in \mathsf{V}$ and by (VAR-MAXTAILS) $\mathsf{max\_tails}_\rho(x, \mathsf{V}) = 0$, hence the thesis is immediate.

(V-VAR) By hypothesis, $\mathsf{V} \vdash \mathsf{Tail}_\rho(x) = s$ with $x \notin \mathsf{V}$ and $x \in dom(\rho)$ and $\mathsf{max\_tails}_\rho(x, \mathsf{V}) = k$. To derive this latter hypothesis we must have applied rule (VAR-MAXTAILS), so we also know that $\mathsf{max\_tails}_\rho(\rho(x), \mathsf{V} \cup \{x\}) = k$. By rule (V-VAR) $\mathsf{V} \cup \{x\} \vdash \mathsf{Tail}_\rho(\rho(x)) = s$ therefore by inductive hypothesis, we have $\mathsf{max\_tails}_\rho(s, \mathsf{V} \cup \{x\}) = k'$ with $k' \leqslant k + 1$, and, thus, the thesis.

(V-NOP) By hypothesis, we have $\mathsf{V} \vdash \mathsf{Tail}_\rho(s_1[nop]s_2) = s_1^\wedge[nop]s_2^\wedge$ and $\mathsf{max\_tails}_\rho(s_1[nop]s_2, \mathsf{V}) = k$. To derive this latter hypothesis we must have applied rule (OP-MAXTAILS), so we also know that $\mathsf{max\_tails}_\rho(s_1, \mathsf{V}) = k_1$ and $\mathsf{max\_tails}_\rho(s_2, \mathsf{V}) = k_2$ with $\max(k_1, k_2) = k$. By rules (TAIL-MAXTAILS) and (OP-MAXTAILS) we have

$$\mathsf{max\_tails}_\rho(s_1^\wedge[nop]s_2^\wedge, \mathsf{V}) = \max(k_1 + 1, k_2 + 1) \leqslant \max(k_1, k_2) + 1 \leqslant k + 1,$$

hence the thesis.

(V-$\|$) By hypothesis, $\mathsf{V} \vdash \mathsf{Tail}_\rho(s_1 \parallel s_2) = s_2 \parallel s_1^\wedge$ and $\mathsf{max\_tails}_\rho(s_1 \parallel s_2, \mathsf{V}) = k$. To derive this latter hypothesis we must have applied rule (OP-MAXTAILS), so we also know that $\mathsf{max\_tails}_\rho(s_1, \mathsf{V}) = k_1$ and $\mathsf{max\_tails}_\rho(s_2, \mathsf{V}) = k_2$ with $\max(k_1, k_2) = k$. By rules (TAIL-MAXTAILS) and (OP-MAXTAILS) we have

$$\mathsf{max\_tails}_\rho(s_2 \parallel s_1^\wedge, \mathsf{V}) = \max(k_2, k_1 + 1) \leqslant \max(k_1, k_2) + 1 \leqslant k + 1,$$

hence the thesis. □

Lemma 5.3. *If* $\mathsf{V} \vdash \mathsf{Tail}_\rho(s_0) = s'$ *and* $\mathsf{bin\_ops}_\rho(s_0, \mathsf{V}) = k$*, then we have* $\mathsf{bin\_ops}_\rho(s', \mathsf{V}) = k'$ *and* $k' \leqslant k$.

Proof. By induction on the rules defining $\mathsf{V} \vdash \mathsf{Tail}_\rho(s_0) = s'$.

(V-TAILS-CONS) By hypothesis, $\mathsf{V} \vdash \mathsf{Tail}_\rho(n : s) = s$ and $\mathsf{bin\_ops}_\rho(n:s, \mathsf{V}) = k$. To derive this latter hypothesis we must have applied rule (CONS-BINOPS), so we also know that $\mathsf{bin\_ops}_\rho(s, \mathsf{V}) = k - 1$. The thesis immediately follows from this fact.

(V-TAIL) By hypothesis, $\mathsf{V} \vdash \mathsf{Tail}_\rho(s^\wedge) = s''$ and $\mathsf{bin\_ops}_\rho(s^\wedge, \mathsf{V}) = k$. To derive this latter hypothesis we must have applied rule (TAIL-BINOPS), so we also know that $\mathsf{bin\_ops}_\rho(s, \mathsf{V}) = k$; furthermore, by rule (V-TAIL), $\mathsf{V} \vdash \mathsf{Tail}_\rho(s) = s'$, therefore by inductive hypothesis, we know that $\mathsf{bin\_ops}_\rho(s', \mathsf{V}) = k'$ with $k' \leqslant k$. Again, by rule (V-TAIL), $\mathsf{V} \vdash \mathsf{Tail}_\rho(s') = s''$, therefore by inductive hypothesis $\mathsf{bin\_ops}_\rho(s'', \mathsf{V}) = k''$ with $k'' \leqslant k' \leqslant k$, hence we get the thesis.

(V-VAR-AX) By hypothesis, $\mathsf{V} \vdash \mathsf{Tail}_\rho(x) = x$ with $x \in \mathsf{V}$ and by (VAR-BINOPS) $\mathsf{bin\_ops}_\rho(x, \mathsf{V}) = 0$, hence the thesis is immediate.

(V-VAR) By hypothesis, $\mathsf{V} \vdash \mathsf{Tail}_\rho(x) = s$ with $x \notin \mathsf{V}$ and $x \in dom(\rho)$ and $\mathsf{bin\_ops}_\rho(x, \mathsf{V}) = k$. To derive this latter hypothesis we must have applied rule (VAR-BINOPS), so we also know that $\mathsf{bin\_ops}_\rho(\rho(x), \mathsf{V} \cup \{x\}) = k$. By rule (V-VAR) $\mathsf{V} \cup \{x\} \vdash \mathsf{Tail}_\rho(\rho(x)) = s$ therefore by inductive hypothesis, we have $\mathsf{bin\_ops}_\rho(s, \mathsf{V} \cup \{x\}) = k'$ with $k' \leqslant k$, and, thus, the thesis.

(V-NOP) By hypothesis, we have $\mathsf{V} \vdash \mathsf{Tail}_\rho(s_1[nop]s_2) = s_1^\wedge[nop]s_2^\wedge$ and $\mathsf{bin\_ops}_\rho(s_1[nop]s_2, \mathsf{V}) = k$. To derive this latter hypothesis we must have applied rule (OP-BINOPS), so we also know that $\mathsf{bin\_ops}_\rho(s_1, \mathsf{V}) = k_1$ and $\mathsf{bin\_ops}_\rho(s_2, \mathsf{V}) = k_2$ with $k = k_1 + k_2 + 1$. By rules (TAIL-BINOPS) and (OP-BINOPS), we have $\mathsf{bin\_ops}_\rho(s_1^\wedge[nop]s_2^\wedge, \mathsf{V}) = k_1 + k_2 \leqslant k_1 + k_2 + 1 = k$, hence the thesis.

(V-‖) By hypothesis, $\mathsf{V} \vdash \mathsf{Tail}_\rho(s_1 \parallel s_2) = s_2 \parallel s_1^\wedge$ and $\mathsf{bin\_ops}_\rho(s_1 \parallel s_2, \mathsf{V}) = k$. To derive this latter hypothesis we must have applied rule (OP-BINOPS), so we also know that $\mathsf{bin\_ops}_\rho(s_1, \mathsf{V}) = k_1$ and $\mathsf{bin\_ops}_\rho(s_2, \mathsf{V}) = k_2$ with $k = k_1 + k_2 + 1$. By rules (TAIL-BINOPS) and (OP-BINOPS), we have $\mathsf{bin\_ops}_\rho(s_2 \parallel s_1^\wedge, \mathsf{V}) = k_1 + k_2 \leqslant k_1 + k_2 + 1 = k$, hence the thesis. □

Lemma 5.4. *If* $\mathsf{Tail}_\rho(x) = s'$ *then rule* (VAR) *cannot be applied for variable* x *in the derivation of* $\mathsf{Tail}_\rho(\rho(x)) = s'$.

Proof. We first observe that for any term s_0 there exists a unique applicable rule for deriving $\mathsf{Tail}_\rho(s_0) = s'$, therefore there cannot exist two different derivations for $\mathsf{Tail}_\rho(s_0) = s'$. From this and the fact that derivations must be finite (because the definition of $\mathsf{Tail}_\rho(s_0) = s'$ is inductive) we can deduce that, for any path in the derivation of $\mathsf{Tail}_\rho(\rho(x)) = s'$, rule (VAR) cannot be applied for the same variable x, otherwise the path would not be finite. □

Lemma 5.5. *Let* V *be a set of variables; if* $\mathsf{Tail}_\rho(s_0) = s'$ *is derivable by applying rule* (VAR) *only for variables not in* V*, then* $\mathsf{V} \vdash \mathsf{Tail}_\rho(s_0) = s'$ *is derivable.*

Proof. The proof proceeds by induction on the rules for $\mathsf{Tail}_\rho(s_0) = s'$; the only non trivial case is for (VAR), that is, when $s_0 = x$. By Lemma 5.4 rule (VAR) cannot be applied for variable x in the derivation of $\mathsf{Tail}_\rho(\rho(x)) = s'$, therefore by inductive hypothesis $\mathsf{V} \cup \{x\} \vdash \mathsf{Tail}_\rho(s_0) = s'$ is derivable, and, hence, we can conclude $\mathsf{V} \vdash \mathsf{Tail}_\rho(x) = s'$ by rule (V-VAR), because by hypothesis $x \notin \mathsf{V}$. □

Corollary 5.6. *If* $\mathsf{Tail}_\rho(s_0) = s'$ *then* $\emptyset \vdash \mathsf{Tail}_\rho(s_0) = s'$.

Proof. A direct consequence of Lemma 5.5. □

Recall that $\mathsf{max\_tails}_\rho(s)$ and $\mathsf{bin\_ops}_\rho(s)$ are defined as $\mathsf{max\_tails}_\rho(s, \emptyset)$ and $\mathsf{bin\_ops}_\rho(s, \emptyset)$, respectively.

Corollary 5.7. *If* $\mathsf{Tail}_\rho(s_0) = s'$ *and* $\mathsf{max\_tails}_\rho(s_0) = k$*, then* $\mathsf{max\_tails}_\rho(s') = k'$ *and* $k' \leqslant k + 1$.

Proof. Directly from Lemma 5.2 and Lemma 5.5. □

Corollary 5.8. *If* $\mathsf{Tail}_\rho(s_0) = s'$ *and* $\mathsf{bin\_ops}_\rho(s_0) = k$*, then* $\mathsf{bin\_ops}_\rho(s') = k'$ *and* $k' \leqslant k$.

Proof. Directly from Lemma 5.3 and Lemma 5.5. □

Definition 5.9. For any s_1, s_2, ρ, the measure of $s_1 \approx_\rho s_2$, denoted by $\mu_\rho(s_1, s_2)$, is defined as follows:

$$\mu_\rho(s_1, s_2) = (k_1, k_2) \text{ where}$$
$$k_1 = \max(\{\mathsf{max\_tails}_\rho(s_i) \mid i = 1, 2\} \cup \{\mathsf{max\_tails}_\rho(\rho(x)) \mid x \in dom(\rho)\})$$
$$k_2 = \max(\{\mathsf{bin\_ops}_\rho(s_i) \mid i = 1, 2\} \cup \{\mathsf{bin\_ops}_\rho(\rho(x)) \mid x \in dom(\rho)\})$$

The following theorem shows that the measure of each node $s_1 \approx_\rho s_2$ in a possibly infinite derivation for $s_0 \approx_\rho s_0'$ is bounded by the measure of the root $s_0 \approx_\rho s_0'$, where componentwise order is considered: $(k_1, k_2) \leqslant (k_1', k_2')$ iff $k_1 \leqslant k_1'$ and $k_2 \leqslant k_2'$.

Theorem 5.10. *If* $s_0 \approx_\rho s_0'$*, then* $\mu_\rho(s_1, s_2) \leqslant \mu_\rho(s_0, s_0')$ *for all* $s_1 \approx_\rho s_2$ *in the derivation of* $s_0 \approx_\rho s_0'$.

Proof. By induction on the length of the path in the derivation from the root $s_0 \approx_\rho s_0'$ to the node $s_1 \approx_\rho s_2$ and by case analysis on the applied rule for the root. For rules (TAIL-L) and (TAIL-R) the claims on $\mathsf{Tail}_\rho(s) = s'$ in Corollary 5.7 and Corollary 5.8 are exploited. □

The following lemma and theorem prove that the set of nodes $s_0 \approx_\rho s_0'$ with measure bounded by a constant is always finite, under the assumption that s_0 and s_0' are built over a fixed finite set of variables.

Lemma 5.11. *Let* V *be a fixed finite set of variables,* ρ *an environment, and* k_1, k_2 *two natural numbers; then the set* $\{s \mid (\mathsf{max\_tails}_\rho(s), \mathsf{bin\_ops}_\rho(s)) = (k_1, k_2), s$ *built on* $\mathsf{V}\}$ *is finite.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{stuck}}{0 : x \approx_\rho (0:x) \parallel x}\,(??)}{0 : x \approx_\rho x}\,(\text{VAR-R}) \qquad \dfrac{\overline{x \approx_\rho x}\,(\text{VAR})}{x \approx_\rho (0:x)^\wedge}\,(\text{TAIL-R})}{(0:x) \parallel x \approx_\rho x \parallel (0:x)^\wedge}\,(\text{OP}) \qquad \dfrac{\overline{\mathsf{Tail}_\rho((0:x) \parallel x) = x \parallel (0:x)^\wedge}\,(\text{TAIL-}\parallel)}{\mathsf{Tail}_\rho(x) = x \parallel (0:x)^\wedge}\,(\text{TAIL-VAR})}{(0:x) \parallel x \approx_\rho x^\wedge}\,(\text{TAIL-R})}{x \approx_\rho x^\wedge}\,(\text{VAR-L})$$

Fig. 10. $x \approx_\rho x^\wedge$ with $\rho = \{x \mapsto (0:x) \parallel x\}$.

Proof. By induction on the well-founded componentwise order on pairs (k_1, k_2).

- $(k_1, k_2) = (0,0)$: if s contains at least a tail operator, then by definition $\mathsf{max\_tails}_\rho(s) > 0$, whereas if it contains at least another operator, then by definition $\mathsf{bin\_ops}_\rho(s) > 0$, therefore s can only be a variable and the thesis follows from the hypothesis on V.
- $(k_1, k_2) > (0,0)$: the proof for the inductive step proceeds by case analysis on the form of s.
  - variable: the set of terms such that s is a variable must be finite, again by hypothesis on V;
  - tail: consider the set $S_1 = \{s \mid (\mathsf{max\_tails}_\rho(s), \mathsf{bin\_ops}_\rho(s)) = (k_1, k_2), s = s_0^\wedge, s \text{ built on } \mathsf{V}\}$; by definition, $\mathsf{max\_tails}_\rho(s_0) = k_1 - 1$ and $\mathsf{bin\_ops}_\rho(s_0) = k_2$, therefore by generalized inductive hypothesis we deduce that the set of terms $\{s_0 \mid (\mathsf{max\_tails}_\rho(s_0), \mathsf{bin\_ops}_\rho(s_0)) = (k_1 - 1, k_2), s_0 \text{ built on } \mathsf{V}\}$ is finite, therefore S_1 is finite as well;
  - constructor: consider the set $S_2 = \{s \mid (\mathsf{max\_tails}_\rho(s), \mathsf{bin\_ops}_\rho(s)) = (k_1, k_2), s = n : s_0, s \text{ built on } \mathsf{V}\}$; by definition, $\mathsf{max\_tails}_\rho(s_0) = k_1$ and $\mathsf{bin\_ops}_\rho(s_0) = k_2 - 1$, therefore we deduce that S_2 is finite similarly as done for the previous case;
  - other binary operators: consider the set $S_3 = \{s \mid (\mathsf{max\_tails}_\rho(s), \mathsf{bin\_ops}_\rho(s)) = (k_1, k_2), s = s_1 \; op \; s_2, s \text{ built on } \mathsf{V}\}$; by definition, $\mathsf{max\_tails}_\rho(s_i) = k_1^i \leqslant k_1$ and $\mathsf{bin\_ops}_\rho(s_i) = k_2^i < k_2$, $i = 1,2$, therefore by generalized inductive hypothesis we deduce that the sets of terms $\{s_i \mid (\mathsf{max\_tails}_\rho(s_i), \mathsf{bin\_ops}_\rho(s_i)) = (k_1^i, k_2^i), s_i \text{ built on } \mathsf{V}\}$, are finite for all $(k_1^i, k_2^i) < (k_1, k_2)$, $i = 1,2$. Since all possible pairs (k_1^i, k_2^i) are finite because bounded by (k_1, k_2) we deduce that the set of all possible subterms s_1, s_2 of s is finite, therefore S_3 is finite as well.

  We have partitioned the set $\{s \mid (\mathsf{max\_tails}_\rho(s), \mathsf{bin\_ops}_\rho(s)) = (k_1, k_2), s \text{ built on } \mathsf{V}\}$ into four sets depending on the form of s and proved that all such sets are finite, therefore we can conclude the claim. □

Theorem 5.12. *Let V be a finite set of variables, ρ an environment, and k_1, k_2 two natural numbers; then the set $\{s_1 \approx_\rho s_2 \mid \mu_\rho(s_1, s_2) \leqslant (k_1, k_2), s_1, s_2$ built on $\mathsf{V}\}$ is finite.*

Proof. A direct consequence of Lemma 5.11, and of the facts that V, the domain of ρ and the set $\{(k_1', k_2') \mid (k_1', k_2') \leqslant (k_1, k_2)\}$ are finite, and the finite union of finite sets is finite. □

Theorem 5.13. *Any derivation for $s_0 \approx_\rho s_0'$ involves only a finite set of pairs $s_1 \approx_\rho s_2$.*

Proof. A direct consequence of Theorem 5.10 and Theorem 5.12. □

6. Conclusion

This paper continues a stream of research [3,4,6] aiming at enhancing regular corecursion beyond regular terms, focusing on the paradigmatic example of streams. The main contribution is that we provide a procedure to check stream equivalence. Notably, we define an equivalence judgment through a coinductive inference system, and prove its soundness and relative completeness. Then, we show that this definition can be turned into an equivalent algorithmic procedure.

Decidability of stream equivalence. We have proved that there is a sound algorithm for stream equivalence in our calculus but, as anticipated, the equivalence judgment we have defined is not complete, since its rules do not consider additional axioms that are expected to hold for the interleaving and the pointwise arithmetic operators.

Completeness of the equivalence check means that we should be able to prove that two streams are equivalent by using the rules in Fig. 5 whenever they define the same abstract stream, that is, access to any index gives the same result.

For instance, if we consider the equivalence $x \approx_\rho x^\wedge$ in the environment $\rho = \{x \mapsto (0:x) \parallel x\}$ we have a stuck derivation, see Fig. 10, even though x and $x^\wedge$ both represent the stream constantly equal to 0 (we have omitted the derivation for $\mathsf{Tail}_\rho(0:x) = x$). It is easy to see that a similar stuck path in the derivation is obtained if rule (TAIL-R) is applied before (VAR-L).

Similarly, our definition of equivalence does not take into account properties of the pointwise arithmetic operators, e.g., it is not possible to derive equivalences as

- $(x \,[+]\, x, \rho) \approx_\rho (y \,[+]\, z, \rho)$, with $\rho = \{x \mapsto 2 : x, y \mapsto 1 : y, z \mapsto 3 : z\}$
- $(x \,[+]\, y, \rho) \approx_\rho (y \,[+]\, x, \rho)$, for all ρ
- $((x \,[+]\, y) \,[+]\, z, \rho) \approx_\rho (x \,[+]\, (y \,[+]\, z), \rho)$, for all ρ

The decidability of the equivalence problem with interleaving and the pointwise arithmetic is an open problem. In literature few results can be found on decidability of stream equivalence.

Grabmayer et al. [16] prove decidability of equivalence between streams defined by well-defined recursive equations as in our calculus, with the limitation that stream constructor and interleaving, a.k.a. *zip*, are the only allowed operators. Their result is generalized to *zip* with any fixed arity $\geqslant 2$, while it remains an open problem whether equivalence is decidable when *zip* with different arities can be mixed together. The same work proves also that equivalence becomes undecidable if generalized projection of streams is added as further operator, thus showing, in a sense, that the decidability result for *zip* is at the border of undecidability. This does not leave much hope that decidability holds when also the tail and the pointwise arithmetic operators of our calculus are considered.

Negative results are proved by Endrullis et al. [14], where more notions of stream equivalence, at different levels of abstraction, and more general systems of equations are considered. They show that, even with the most relaxed models, not only equivalence is undecidable, but also there is no proof system that is complete in general. This does not come to surprise, given the general type of systems of equations considered by the authors: equations are allowed to admit no unique solution and can be defined over any Σ-algebra which includes the *head* and *tail* operators.

Stream calculi. In the context of streams, our main sources of inspiration for the operators considered in the calculus and some examples have been the works of Rutten [24,25], where a coinductive calculus of streams of real numbers is defined, and Hinze [17], where a calculus of generic streams is defined in a constructive way and implemented in Haskell. The work presented here differs from such approaches since, in our case, the aim is to provide an *operational* semantics, designed to directly lead to an implementation. That is, even though streams are infinite objects (terms where the constructor is the only operator, defined coinductively), evaluation handles *finite* representations, and is defined by an *inductive* inference system.

Another work to be mentioned is LOLA [13], a specification language for runtime monitoring that manipulates streams. The general idea behind the framework is to generate a set of output streams, starting from a given set of input streams. Input streams describe the values of the system under observation and output streams represent errors or reports of the monitor. Stream expressions are constructed starting from constants, stream variables, function symbols and, in particular, an operator to define a stream by starting from another one shifted by an offset; in this case, a default value must be specified because the offset might lead past the end or before the beginning of the stream. The main difference with respect to our work is that LOLA only allows streams with a finite number of elements.

Beyond regular terms. The main disadvantage of regular corecursion with respect to lazy evaluation is that only regular structures are supported. For instance, given the function definition `from(n) = n : from(n+1)`, a call like, e.g., `member(10,from(0))` will not terminate in our calculus, since with call-by-value semantics the subterm `from(0)` should be evaluated. There have been a few proposals to allow the manipulation of infinite non-regular values. Notably, in the context of logic programming, *structural resolution* [21,20] (a.k.a. S-resolution) is a proposed generalization for cases when formulas computable at infinity are not regular; infinite derivations that cannot be built in finite time are generated lazily, and only partial answers are shown. In particular, recent results [7] investigate how it is possible to integrate co-LP cycle detection into S-resolution, by proposing a comprehensive theory to provide operational semantics that go beyond loop detection.

Another approach is the work by Courcelle [9], introducing algebraic trees and equations as generalizations of regular ones. Such proposals share, even though with different techniques and in a different context, our aim of extending regular corecursion.

Future work. A first objective for future work is to enrich the language by adding more data types and constructs, e.g., an `if-then-else` with a stream of booleans as condition and two streams in the branches. Additional interesting operators on streams could be added, such as the filtering ones, which would greatly increase the expressive power of the calculus. Each new operator should be thoroughly studied, in order to be dealt with by equivalence checks, as done here for the interleaving operator. Another interesting direction is the design of a static type system to filter out early errors, such as a static check filtering out certain non-well-defined streams beforehand.

A possible direction to optimize the semantics could be to handle regular streams in a special way. Indeed, such kind of streams can be finitely represented by their first few digits and period, and with just this information we can compute functions like *at* in constant time. To achieve this goal, the semantics should be able to identify regular streams and treat them differently, for example by applying special rules designed for optimizing performance.

CRediT authorship contribution statement

Davide Ancona: Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Pietro Barbieri:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Elena Zucca:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

$$
\begin{array}{lcll}
s & ::= & x \mid n : s \mid s^{\wedge} \mid s_1\ op\ s_2 & \text{(open) stream term} \\
op & ::= & [nop] \mid \| & \text{binary stream operator} \\
\rho & ::= & x_1 \mapsto s_1 \dots x_n \mapsto s_n \quad (n \geqslant 0) & \text{environment} \\
\varepsilon & ::= & (s_1, s_1') \dots (s_n, s_n') \quad (n \geqslant 0) & \text{equivalence trace}
\end{array}
$$

$$
(\textsc{corec})\ \frac{}{\varepsilon \vdash s_1 \approx_\rho s_2}\ (s_1, s_2) \in \varepsilon
$$

$$
(\textsc{var-l})\ \frac{\varepsilon \cdot (x, s) \vdash \rho(x) \approx_\rho s}{\varepsilon \vdash x \approx_\rho s} \qquad (\textsc{var-r})\ \frac{\varepsilon \cdot (s, x) \vdash s \approx_\rho \rho(x)}{\varepsilon \vdash s \approx_\rho x} \qquad (\textsc{var})\ \frac{}{\varepsilon \vdash x \approx_\rho x}
$$

$$
(\textsc{cons})\ \frac{\varepsilon \vdash s_1 \approx_\rho s_2}{\varepsilon \vdash n : s_1 \approx_\rho n : s_2}
$$

$$
(\textsc{tail-l})\ \frac{\varepsilon \vdash s' \approx_\rho s_2}{\varepsilon \vdash s_1{}^{\wedge} \approx_\rho s_2}\ \mathsf{Tail}_\rho(s_1) = s' \qquad (\textsc{tail-r})\ \frac{\varepsilon \vdash s_1 \approx_\rho s'}{\varepsilon \vdash s_1 \approx_\rho s_2{}^{\wedge}}\ \mathsf{Tail}_\rho(s_2) = s'
$$

$$
(\textsc{op})\ \frac{\varepsilon \vdash s_1 \approx_\rho s_1' \qquad \varepsilon \vdash s_2 \approx_\rho s_2'}{\varepsilon \vdash s_1\ op\ s_2 \approx_\rho s_1'\ op\ s_2'}
$$

Fig. A.11. Equivalence check with trace.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{(x, y^{\wedge\wedge})\} \vdash x \approx_\rho y^{\wedge\wedge}}\ (\textsc{corec})}{\{(x, y^{\wedge\wedge})\} \vdash 1 : x \approx_\rho 1 : y^{\wedge\wedge}}\ (\textsc{cons}) \qquad \mathsf{Tail}_\rho(y^{\wedge}) = 1 : y^{\wedge\wedge}}{\{(x, y^{\wedge\wedge})\} \vdash 1 : x \approx_\rho y^{\wedge\wedge}}\ (\textsc{tail-r})}{\vdash x \approx_\rho y^{\wedge\wedge}}\ (\textsc{var-l})
$$

$$
\dfrac{\dfrac{\dfrac{}{\mathsf{Tail}_\rho(2 : 3 : 1 : y^{\wedge\wedge}) = 3 : 1 : y^{\wedge\wedge}}\ (\textsc{tail-cons})}{\mathsf{Tail}_\rho(y) = 3 : 1 : y^{\wedge\wedge}}\ (\textsc{tail-var}) \qquad \dfrac{}{\mathsf{Tail}_\rho(3 : 1 : y^{\wedge\wedge}) = 1 : y^{\wedge\wedge}}\ (\textsc{tail-cons})}{\mathsf{Tail}_\rho(y^{\wedge}) = 1 : y^{\wedge\wedge}}\ (\textsc{tail-tail})
$$

Fig. A.12. $\vdash x \approx_\rho y^{\wedge\wedge}$ with $\rho = \{x \mapsto 1 : x, y \mapsto 2 : 3 : 1 : y^{\wedge\wedge}\}$.

Acknowledgements

This work was partially funded by the MUR project "T-LADIES" (PRIN 2020TL3X8X). We thank the anonymous reviewers of ICTCS'22 and TCS for their useful comments.

Appendix A. Equivalence check with trace

The judgment $\varepsilon \vdash s_1 \approx_\rho s_2$, defined in Fig. A.11, means that s_1 and s_2 are equivalent in ρ under the *equivalence trace* ε, containing coinductive hypotheses as pairs of already considered stream terms. The trace is abstractly considered as a set, so order and repetitions are immaterial.

The rules are analogous to those in Fig. 5, 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 terms are equivalent if the same pair is contained in the trace of coinductive hypotheses, that is, has been already considered.

Whereas the judgment $s_1 \approx_\rho s_2$ in Fig. 5 is defined coinductively (that is, infinite derivations are allowed), the judgment $\varepsilon \vdash s_1 \approx_\rho s_2$ is defined inductively (only finite derivations are considered). 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 [11].

In Fig. A.12 and Fig. A.13 we report the derivations for the examples of Sect. 5 using the judgment with the equivalence trace. They are exactly the same, with only two differences: a new pair is added to the equivalence trace ε whenever a rule for variables is applied, and rule (COREC) is applied to end the derivation, as soon as an already processed pair is found.

In Fig. A.14, we present an additional example showing the usage of rule (VAR). We consider the derivation of the equivalence $x \approx_\rho y^{\wedge}$ in the environment $\rho = \{x \mapsto 5 : z, y \mapsto 3 : 5 : x^{\wedge}, z \mapsto 2 : 5 : z\}$.

Here we apply (VAR-L) as first rule to retrieve the value of x. Then, the tail of y is computed, and we omit the derivation since it is easy to see that $y^{\wedge} = (3 : 5 : x^{\wedge})^{\wedge} = 5 : x^{\wedge}$. The derivation proceeds with the application of rule (CONS) and, after that, the tail of x is computed. At this point we can apply rule (VAR) to conclude the derivation since we found two equal variables.

In Fig. A.15 we report the derivation for these examples in the coinductive case to show that, thanks to rule (VAR), we can compute the result within a finite number of steps.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{(x,y),(1:(x\,[+]\,x),y)\}\vdash x\approx_\rho y}\,(\text{COREC})}{\{(x,y),(1:(x\,[+]\,x),y)\}\vdash x\approx_\rho (1:y)^\wedge}\,(\text{TAIL-R}) \quad \dfrac{\dfrac{}{\{(x,y),(1:(x\,[+]\,x),y)\}\vdash x\approx_\rho y}\,(\text{COREC})}{\{(x,y),(1:(x\,[+]\,x),y)\}\vdash x\approx_\rho (1:y)^\wedge}\,(\text{TAIL-R})}{\{(x,y),(1:(x\,[+]\,x),y)\}\vdash x\,[+]\,x\approx_\rho (1:y)^\wedge\,[+]\,(1:y)^\wedge}\,(\text{OP}) \quad T_1}{\{(x,y),(1:(x\,[+]\,x),y)\}\vdash x\,[+]\,x\approx_\rho ((1:y)\,[+]\,(1:y))^\wedge}\,(\text{TAIL-R})}{\{(x,y),(1:(x\,[+]\,x),y)\}\vdash 1:(x\,[+]\,x)\approx_\rho 1:((1:y)\,[+]\,(1:y))^\wedge}\,(\text{CONS})}{\{(x,y)\}\vdash 1:(x\,[+]\,x)\approx_\rho y}\,(\text{VAR-R})}{\vdash x\approx_\rho y}\,(\text{VAR-L})$$

$$T_1 = \dfrac{}{\mathsf{Tail}_\rho((1:y)\,[+]\,(1:y)) = (1:y)^\wedge\,[+]\,(1:y)^\wedge}\,(\text{TAIL-NOP})$$

Fig. A.13. $\vdash x\approx_\rho y$ with $\rho = \{x\mapsto 1:(x\,[+]\,x), y\mapsto 1:((1:y)\,[+]\,(1:y))^\wedge\}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{(x,y^\wedge)\}\vdash z\approx_\rho z}\,(\text{VAR}) \quad \mathsf{Tail}_\rho(x) = z}{\{(x,y^\wedge)\}\vdash z\approx_\rho x^\wedge}\,(\text{TAIL-R})}{\{(x,y^\wedge)\}\vdash 5:z\approx_\rho 5:x^\wedge}\,(\text{CONS}) \quad \mathsf{Tail}_\rho(y) = 5:x^\wedge}{\{(x,y^\wedge)\}\vdash 5:z\approx_\rho y^\wedge}\,(\text{TAIL-R})}{\vdash x\approx_\rho y^\wedge}\,(\text{VAR-L})$$

Fig. A.14. $\vdash x\approx_\rho y^\wedge$ with $\rho = \{x\mapsto 5:z, y\mapsto 3:5:x^\wedge, z\mapsto 2:5:z\}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{z\approx_\rho z}\,(\text{VAR})}{z\approx_\rho x^\wedge}\,(\text{TAIL-R}) \quad \mathsf{Tail}_\rho(x) = z}{5:z\approx_\rho 5:x^\wedge}\,(\text{CONS}) \quad \mathsf{Tail}_\rho(y) = 5:x^\wedge}{5:z\approx_\rho y^\wedge}\,(\text{TAIL-R})}{x\approx_\rho y^\wedge}\,(\text{VAR-L})$$

Fig. A.15. $x\approx_\rho y^\wedge$ with $\rho = \{x\mapsto 5:z, y\mapsto 3:5:x^\wedge, z\mapsto 2:5:z\}$.

References

[1] Jirí Adámek, Stefan Milius, Jiri Velebil, Free iterative theories: a coalgebraic view, Math. Struct. Comput. Sci. 13 (2) (2003) 259–320.
[2] Davide Ancona, Pietro Barbieri, Francesco Dagnino, Elena Zucca, Sound regular corecursion in coFJ, in: Robert Hirschfeld, Tobias Pape (Eds.), ECOOP'20 - Object-Oriented Programming, in: LIPIcs, vol. 166, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 1:1–1:28.
[3] Davide Ancona, Pietro Barbieri, Elena Zucca, Enhanced regular corecursion for data streams, in: Claudio Sacerdoti Coen, Ivano Salvo (Eds.), ICTCS'21 - Italian Conf. on Theoretical Computer Science, in: CEUR Workshop Proceedings, vol. 3072, CEUR-WS.org, 2021, pp. 266–280.
[4] Davide Ancona, Pietro Barbieri, Elena Zucca, Enhancing expressivity of checked corecursive streams, in: Michael Hanus, Atsushi Igarashi (Eds.), FLOPS 2022 - Functional and Logic Programming, in: Lecture Notes in Computer Science, vol. 13215, Springer, 2022, pp. 1–18.
[5] Davide Ancona, Pietro Barbieri, Elena Zucca, Equality of corecursive streams defined by finitary equational systems, in: Ugo Dal Lago, Daniele Gorla (Eds.), ICTCS'22 - Italian Conf. on Theoretical Computer Science, in: CEUR Workshop Proceedings, vol. 3284, CEUR-WS.org, 2022, pp. 86–98.
[6] Davide Ancona, Pietro Barbieri, Elena Zucca, Checked corecursive streams: expressivity and completeness, Theor. Comput. Sci. 974 (2023) 114081.
[7] Henning Basold, Ekaterina Komendantskaya, Yue Li, Coinduction in uniform: foundations for corecursive proof search with Horn clauses, in: European Symposium on Programming, ESOP 2019, 2019, pp. 783–813.
[8] Yves Bertot, Ekaterina Komendantskaya, Using structural recursion for corecursion, in: Stefano Berardi, Ferruccio Damiani, Ugo de'Liguoro (Eds.), Types for Proofs and Programs, Springer, 2009, pp. 220–236.
[9] Bruno Courcelle, Fundamental properties of infinite trees, Theor. Comput. Sci. 25 (1983) 95–169.
[10] Guy Cousineau, An algebraic definition for control structures, Theor. Comput. Sci. 12 (2) (1980) 175–192.
[11] Francesco Dagnino, Foundations of regular coinduction, Log. Methods Comput. Sci. 17 (4) (2021).
[12] Francesco Dagnino, Davide Ancona, Elena Zucca, Flexible coinductive logic programming, Theory Pract. Log. Program. 20 (6) (2020) 818–833. Issue for ICLP 2020.
[13] Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, Zohar Manna, LOLA: runtime monitoring of synchronous systems, in: 12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 2005, pp. 166–174.
[14] Jörg Endrullis, Dimitri Hendriks, Rena Bakhshi, Grigore Rosu, On the complexity of stream equality, J. Funct. Program. 24 (2–3) (2014) 166–217.
[15] Vladimir Gapeyev, Michael Y. Levin, Benjamin C. Pierce, Recursive subtyping revealed, J. Funct. Program. 12 (6) (2002) 511–548.
[16] Clemens Grabmayer, Jörg Endrullis, Dimitri Hendriks, Jan Willem Klop, Lawrence S. Moss, Automatic sequences and zip-specifications, in: LICS 2012 - Logic in Computer Science, 2012, pp. 335–344.
[17] Ralf Hinze, Concrete stream calculus: an extended study, J. Funct. Program. 20 (5–6) (2010) 463–535.
[18] Jeannin Jean-Baptiste, Dexter Kozen, Computing with capsules, J. Autom. Lang. Comb. 17 (2–4) (2012) 185–204.
[19] Jean-Baptiste Jeannin, Dexter Kozen, Alexandra Silva, CoCaml: functional programming with regular coinductive types, Fundam. Inform. 150 (2017) 347–377.
[20] Ekaterina Komendantskaya, Patricia Johann, Martin Schmidt, A productivity checker for logic programming, in: Manuel V. Hermenegildo, Pedro López-García (Eds.), Logic-Based Program Synthesis and Transformation - LOPSTR 2016, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 10184, Springer, 2016, pp. 168–186.
[21] Ekaterina Komendantskaya, John Power, Martin Schmidt, Coalgebraic logic programming: from semantics to implementation, J. Log. Comput. 26 (2) (2016) 745–783.
[22] X. Leroy, H. Grall, Coinductive big-step operational semantics, Inf. Comput. 207 (2) (2009) 284–304.

[23] Robin Milner, Mads Tofte, Co-induction in relational semantics, Theor. Comput. Sci. 87 (1) (1991) 209–220.
[24] Jan J.M.M. Rutten, A coinductive calculus of streams, Math. Struct. Comput. Sci. 15 (1) (2005) 93–147.
[25] Jan J.M.M. Rutten, The Method of Coalgebra: Exercises in Coinduction, CWI, February 2019.
[26] Luke Simon, Extending logic programming with coinduction, PhD thesis, University of Texas at Dallas, 2006.
[27] Luke Simon, Ajay Bansal, Ajay Mallya, Gopal Gupta, Co-logic programming: extending logic programming with coinduction, in: Lars Arge, Christian Cachin, Tomasz Jurdzinski, Andrzej Tarlecki (Eds.), Automata, Languages and Programming, ICALP 2007, in: Lecture Notes in Computer Science, vol. 4596, Springer, 2007, pp. 472–483.