Proving Equality of Streams Automatically

Hans Zantema[1] and Jörg Endrullis[2]

1 **Department of Computer Science, TU Eindhoven, The Netherlands**
Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands
h.zantema@tue.nl

2 **Free University Amsterdam, The Netherlands**
joerg@few.vu.nl

Abstract

Streams are infinite sequences over a given data type. A stream specification is a set of equations intended to define a stream. In this paper we focus on equality of streams, more precisely, for a given set of equations two stream terms are said to be equal if they are equal in every model satisfying the given equations. We investigate techniques for proving equality of streams suitable for automation. Apart from techniques that were already available in the tool CIRC from Lucanu and Roşu, we also exploit well-definedness of streams, typically proved by proving productivity. Moreover, our approach does not restrict to behavioral input format and does not require termination.

We present a tool Streambox that can prove equality of a wide range of examples fully automatically.

Digital Object Identifier 10.4230/LIPIcs.RTA.2011.393

Category Regular Research Paper

1 Introduction

The stream ones specified by ones = 1 : ones is the infinite stream having the data element 1 on every position. The same holds for the stream c specified by $c = 1 : 1 : c$. So we expect that $c =$ ones holds. This paper is about how to prove such stream equalities fully automatically. More precisely, given a set of stream equalities (in this case ones = 1 : ones and $c = 1 : 1 : c$) that are assumed to hold, can we conclude validity of another given stream equality (in this case $c =$ ones)? The semantics of this question is quite natural: we have a basic data type D, typically the booleans or natural numbers, and streams are maps from natural numbers to D. The question now states whether if all constants and functions occurring in the equalities are interpreted by streams and stream functions in such a way that all given equalities hold, then also the goal equality holds.

An excellent basis for treating this problem is the circular co-induction principle as presented in [12, 5, 9, 10, 13, 7], and implemented in the tool CIRC. Indeed, several interesting stream equalities can be proved fully automatically by CIRC.

Both the basic circular co-induction principle and its extension using special contexts are the basis of the current paper. However, we offer several features that are not covered by CIRC and/or its underlying theory:

- We do not require the very specific format of behavioral equations in which the root of every left hand side should be hd or tl and the given equations should be terminating when applied only from left to right. Instead we allow any set of equations on streams. Our default format is the pure stream format, the format as used in [3, 14, 15]. In this format

22nd International Conference on Rewriting Techniques and Applications (RTA'11).
Editor: M. Schmidt-Schauß; pp. 393–408
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ones is specified by the single equation ones = 1 : ones, where the similar specification in behavioral format consists of the two equations hd(ones) = 1 and tl(ones) = ones.

- We have a simple syntactic criterion for checking for special contexts: we introduce *guardedness* (Definition 4.4) and show that all contexts composed from symbols satisfying this guardedness, are special.
- In case an equality proof requires auxiliary lemmas, we do not need to enter these lemmas ourselves, but our tool Streambox generates and uses suitable lemmas fully automatically.
- If a particular stream is uniquely defined by a set of equations, and some term satisfies these equations, we may conclude that that term is equal to that stream. In this way we may exploit earlier techniques to prove that a stream is uniquely defined, for instance by proving productivity. In case this is appropriate, our tool Streambox calls a current tool for proving context-sensitive termination for proving productivity, typically succeeding for equations that are non-terminating themselves. In this way state-of-the-art termination tools are exploited for proving stream equality.

Combining these features, Streambox can prove several non-trivial equalities fully automatically, like the property that the Toeplitz stream is the boolean derivative of the Thue-Morse stream, as we shall see in Example 6.4.

The tool CIRC is based on the rewriting engine Maude, in particular using rewriting to normal form as the mechanism to check convertibility. However, for this mechanism the equations are only used in one direction, and the approach typically fails if this rewriting is non-terminating.

Only rewriting in one direction gives undesirable restrictions. For instance, if a stream function f is defined by $f(\sigma) = 0 : f(\sigma)$, and a stream a is defined by $a = f(a)$, then a will coincide with $0 : a$ as is very easily shown by the following conversion:

$$a = f(a) = 0 : f(a) = 0 : a.$$

However, in this conversion the equality $a = f(a)$ is used in both directions, so plain rewriting fails to derive this very simple equality. Polishing the rules, for instance by completion, could be helpful but also has limitations. Applying the rule $a \to f(a)$ will cause non-termination, and $f(a) \to a$ will cause critical pairs with the defining rule for f. In this very simple example Knuth-Bendix completion of the equations will succeed, but for many other examples, for instance involving commutativity, this does not hold. Moreover, in a typical scenario the set of equations for which convertibility of two terms has to be tested is not one static set, but is extended all time by co-induction hypotheses and proved goals and lemmas, for which ongoing Knuth-Bendix completion would be problematic.

So instead of forcing to do rewriting in only one direction and requiring termination, a crucial building block of our approach is a check for general convertibility without these restrictions. Accordingly, we developed our own tooling Streambox from scratch.

A typical example for which our tool Streambox succeeds is in proving $M = 0 : M$, where M is specified by both $M = f(M)$ and $M = g(M)$, for f and g defined mutually recursively by

$$f(x : \sigma) = 0 : g(\sigma), \quad g(x : \sigma) = x : f(\sigma).$$

Here x is a boolean variable and σ is a stream variable. Indeed, f replaces every symbol on an even position by 0, and g does the same for odd positions, so if $M = f(M)$ and $M = g(M)$ then indeed M has to consist only of zeros, by which $M = 0 : M$ should hold. Our goal is that such a property can be proved fully automatically after entering only these equations. Extensive experiments show failure of CIRC on all variants we could think of:

keeping the M-rules as they are will yield non-termination, while reversing them yields non-unique normal forms caused by combinations of rules like $f(M) = M$ and $\mathsf{hd}(f(\sigma)) = 0$.

Conceptually, our notions of circular co-induction and the use of special contexts is the same as in [9, 10, 13, 7]. However, we do not restrict to the behavioral format as required there, by which the underlying definitions look quite different. As streams have a natural semantics as mappings from natural numbers to data, it is natural to relate the notions of validity to this semantics. We succeeded in presenting the theory for arbitrary sets of stream equations based on this semantics. In this paper both the validity of the basic circular co-induction principle (Theorem 3.1) and its extension to special contexts (Theorem 4.2) is proved directly by induction on natural numbers, making the theory self-contained and independent of [9, 10, 13, 7] or any theory of co-induction. More abstract variants of Theorem 3.1 and Theorem 4.2 are given in [10, 13]; there they are proved in a more abstract setting in which it is left implicit that streams are a valid instance.

Most of our theory easily generalizes to other infinite data types, like infinite binary trees. However, in order to keep notations simple we chose to focus on streams. It turns out that involved underlying proof principles already appear in streams, and can be motivated by small boolean stream examples. Our theory is given for streams over arbitrary data types (specified by finite constructor ground terms). In the implementation and most of the examples we restrict to the boolean case in which 0 and 1 are the only data elements. Example 3.4 shows how our theory can be applied for stream over natural numbers.

This paper is structured as follows. In Section 2 we present the basic setting and its semantics. In Section 3 we present the basic version of the principal of circular co-induction. In Section 4 we extend this to special contexts and investigate how to recognize special contexts when given in pure stream format, the standard format as used in [3, 14, 15]. In Section 5 we present how to prove equality by using that a stream or stream function is uniquely defined, typically to be proved by proving productivity. Next, in Section 6 we discuss our implementation. We conclude in Section 7.

2 Streams: Specifications and Models

In stream specifications we have two sorts: s (stream) and d (data). We assume the set D of data elements to consist of ground terms $\mathbf{T}(\Sigma_d)$ over some signature Σ_d of which all symbols are of type $d^n \to d$ for some $n \geq 0$. We will focus on the boolean case where $D = \Sigma_d = \{0, 1\}$. We assume three particular symbols:

- : having type $d \times s \to s$;
- hd having type $s \to d$;
- tl having type $s \to s$.

Apart from the symbols in $\Sigma_d \cup \{:, \mathsf{hd}, \mathsf{tl}\}$ there is a set Σ of user defined symbols, each being of type $d^n \times s^m \to s$ or $d^n \times s^m \to d$ for $n, m \geq 0$.

We assume a set $\mathbf{X}_s$ of variables of sort s and a set $\mathbf{X}_d$ of variables of sort d. Using all these ingredients we can build (well-sorted) terms:

- x is a term of sort d if $x \in \mathbf{X}_d$,
- σ is a term of sort s if $\sigma \in \mathbf{X}_s$,
- if $u_1 \ldots, u_n$ are terms of sort d, and $t_1 \ldots, t_m$ of sort s, then $f(u_1 \ldots, u_n, t_1 \ldots, t_m)$ is a term of sort d if $f \in \Sigma \cup \Sigma_d \cup \{\mathsf{hd}\}$ has type $d^n \times s^m \to d$, and a term of sort s if $f \in \Sigma \cup \{:, \mathsf{tl}\}$ has type $d^n \times s^m \to s$.

The set of all such terms is denoted by $\mathbf{T}(\Sigma \cup \Sigma_d \cup \{:, \mathsf{hd}, \mathsf{tl}\}, \mathbf{X}_d, \mathbf{X}_s)$.

As a notational convention variables of sort d will be denoted by x, y, terms of sort d by u, u_i, variables of sort s by σ, τ, and terms of sort s by t, t_i.

▶ **Definition 2.1.** An *equation* is a pair $(\ell, r) \in \mathbf{T}(\Sigma \cup \Sigma_d \cup \{:, \mathsf{hd}, \mathsf{tl}\}, \mathbf{X}_d, \mathbf{X}_s)^2$ such that ℓ and r are of the same sort.

An equation (ℓ, r) is usually written as $\ell = r$.

A *stream specification* is defined to be a set of equations.

▶ **Example 2.2.** For specifying the Thue-Morse stream morse as extensively studied e.g. in [1], we have $D = \Sigma_d = \{0, 1\}$. There are two equations of sort d:

$$\mathsf{not}(0) = 1, \quad \mathsf{not}(1) = 0,$$

and the following equations of sort s:

$$\begin{array}{rclcrcl} \mathsf{morse} & = & 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{morse}), \mathsf{tl}(\mathsf{morse})) & \qquad & \mathsf{tl}(x : \sigma) & = & \sigma \\ \mathsf{inv}(x : \sigma) & = & \mathsf{not}(x) : \mathsf{inv}(\sigma) & & \mathsf{zip}(x : \sigma, \tau) & = & x : \mathsf{zip}(\tau, \sigma) \end{array}$$

Stream specifications are intended to specify streams for the constants in Σ of sort s, and stream functions for the other elements of Σ of sort s. Similarly, elements of Σ of sort d should specify data elements and data functions. The combination of these streams and functions is what we will call a *stream model*.

More precisely, a *stream* over D is a map from the natural numbers to D. Write $S = D^\omega$ for the set of all streams over D. In case of $D = \emptyset$ we have $S = \emptyset$; in case of $\#D = 1$ we have $\#S = 1$. So in non-degenerate cases we have $\#D \geq 2$.

▶ **Definition 2.3.** A *stream model* over $\Sigma_d \cup \Sigma$ is defined to consist of the set $S = D^\omega$ and a set of functions and constants $[f]$ for every $f \in \Sigma$, where $[f] : D^n \times S^m \to S$ if the type of $f \in \Sigma$ is $d^n \times s^m \to s$, and $[f] : D^n \times S^m \to D$ if the type of $f \in \Sigma$ is $d^n \times s^m \to d$.

Apart from the functions and constants $[f]$ for every $f \in \Sigma$, we also have $[f]$ for every $f \in \Sigma_d \cup \{:, \mathsf{hd}, \mathsf{tl}\}$, defined as follows:

- $[f](u_1, \ldots, u_n) = f(u_1, \ldots, u_n)$ for $f \in \Sigma_d$ of type $d^n \to d$, $u_1, \ldots, u_n \in \mathbf{T}(\Sigma_d)$, for $n \geq 0$.
- For $u \in \mathbf{T}(\Sigma_d)$ and $s \in S$ the stream $[:](u, s)$ is defined by

  $$[:](u, s)(0) = u, \quad [:](u, s)(n + 1) = s(n)$$

  for every $n \geq 0$.
- $[\mathsf{hd}](s) = s(0)$ for $s \in S$.
- For $s \in S$ the stream $[\mathsf{tl}](s)$ is defined by

  $$[\mathsf{tl}](s)(n) = s(n + 1)$$

  for every $n \geq 0$.

A *variable assignment* α is defined to be a map $\alpha : \mathbf{X}_d \cup \mathbf{X}_s \to D \cup S$ such that $\alpha(x) \in D$ for $x \in \mathbf{X}_d$ and $\alpha(\sigma) \in S$ for $\sigma \in \mathbf{X}_s$.

For $t \in \mathbf{T}(\Sigma \cup \Sigma_d \cup \{:, \mathsf{hd}, \mathsf{tl}\}, \mathbf{X}_d, \mathbf{X}_s)$ and a variable assignments $\alpha : \mathbf{X}_d \cup \mathbf{X}_s \to D \cup S$ the *stream interpretation* $[t, \alpha]$ in the stream model $(S, ([f])_{f \in \Sigma})$ is defined inductively by:

$$\begin{array}{rcll} [x, \alpha] & = & \alpha(x) & \text{for } x \in \mathbf{X}_d \\ [\sigma, \alpha] & = & \alpha(\sigma) & \text{for } \sigma \in \mathbf{X}_s \\ [f(u_1, \ldots, u_n, t_1, \ldots, t_m), \alpha] & = & [f]([u_1, \alpha], \ldots, [u_n, \alpha], [t_1, \alpha], \ldots, [t_m, \alpha]) & \end{array}$$

for all terms $u_1, \ldots, u_n$ of sort d and all terms $t_1, \ldots, t_m$ of sort s, and every $f \in \Sigma \cup \Sigma_d \cup \{:, \mathsf{hd}, \mathsf{tl}\}$ of type $d^n \times s^m \to d$ or $d^n \times s^m \to s$.

In case t is a ground term then $[t, \alpha]$ does not depend on α, and we simply write $[t]$ rather than $[t, \alpha]$.

▶ **Definition 2.4.** An equation $\ell = r$ is defined to *hold* in the stream model, or the stream model *satisfies* the equation $\ell = r$, if $[\ell, \alpha] = [r, \alpha]$ for every variable assignment α. A stream model is said to *satisfy a set of equations* if it satisfies each of its equations.

So summarizing: a stream model consists of streams, and interpretations of function symbols as functions on streams. For the symbols $:, \mathsf{hd}, \mathsf{tl}$ the interpretations are predefined; for the other symbols the choice is free. As the interpretations of $:, \mathsf{hd}, \mathsf{tl}$ are fixed it is expected that some corresponding equations hold in every model. Indeed this is the case. Let $\mathbf{E}_{ht}$ be the set of equations:

$$\mathbf{E}_{ht} = \left\{ \begin{array}{rcl} \mathsf{hd}(x : \sigma) & = & x \\ \mathsf{tl}(x : \sigma) & = & \sigma \\ \mathsf{hd}(\sigma) : \mathsf{tl}(\sigma) & = & \sigma \end{array} \right.$$

▶ **Lemma 2.5.** *Every stream model satisfies* $\mathbf{E}_{ht}$.

Proof. We have to check that all three equations of $\mathbf{E}_{ht}$ hold in every stream model. Using the definitions of $[:], [\mathsf{hd}], [\mathsf{tl}]$ we obtain

$$[\mathsf{hd}(x : \sigma), \alpha] = [\mathsf{hd}]([x : \sigma, \alpha]) = [x : \sigma, \alpha](0)[:]([x, \alpha], [\sigma, \alpha])(0) = [x, \alpha],$$

proving that the first equation holds, and

$$[\mathsf{tl}(x : \sigma), \alpha](i) = [\mathsf{tl}]([x : \sigma, \alpha])(i) = [x : \sigma, \alpha](i + 1) = [:]([x, \alpha], [\sigma, \alpha])(i + 1) = [\sigma, \alpha](i)$$

for every $i \geq 0$, proving that the second equation holds. For the last equation we have to prove that $[\mathsf{hd}(\sigma) : \mathsf{tl}(\sigma), \alpha](i) = [\sigma, \alpha](i)$ for every $i \geq 0$. For $i = 0$ this holds since

$$[\mathsf{hd}(\sigma) : \mathsf{tl}(\sigma), \alpha](0) = [:]([\mathsf{hd}(\sigma), \alpha], [\mathsf{tl}(\sigma), \alpha])(0) = [\mathsf{hd}(\sigma), \alpha] = [\sigma, \alpha](0);$$

for $i > 0$ this holds since

$$[\mathsf{hd}(\sigma) : \mathsf{tl}(\sigma), \alpha](i) = [:]([\mathsf{hd}(\sigma), \alpha], [\mathsf{tl}(\sigma), \alpha])(i) = [\mathsf{tl}(\sigma), \alpha](i - 1) = [\sigma, \alpha](i).$$

◀

Now we arrive at the central notion of this paper.

▶ **Definition 2.6.** An equation $\ell = r$ is defined to *hold* with respect to a stream specification E, notation $E \models \ell = r$, if $\ell = r$ holds in every stream model satisfying E.

A set E' of equations is defined to *hold* with respect to a stream specification E, notation $E \models E'$, if $E \models \ell = r$ for every $\ell = r \in E'$.

For a stream specification E we write $=_E$ for the congruence generated by E, that is, the closure of E with respect to reflexivity, symmetry, transitivity, contexts and substitution. Only using $=_E$ is called *equational reasoning*.

We obviously have the following theorem.

▶ **Theorem 2.7.** *For every equation* $\ell = r$ *satisfying* $\ell =_{E \cup \mathbf{E}_{ht}} r$ *we have* $E \models \ell = r$.

The converse of Theorem 2.7 is not true: typically $E \models \ell = r$ may hold while $\ell =_{E \cup \mathbf{E}_{ht}} r$ does not hold. For instance, if E consists of the two equations $c = 0 : c$ and $d = 0 : d$, then in every model both $[c]$ and $[d]$ are equal to stream only consisting of zeros, so $E \models c = d$. However, to prove this equational reasoning is not sufficient: $c =_{E \cup \mathbf{E}_{ht}} d$ does not hold.

3 Basic circular co-induction

A main goal of this paper is to extend Theorem 2.7 to more powerful syntactic techniques suitable for implementation for concluding $E \models \ell = r$, which are still based on equational reasoning. As long as the elements of the intended models can be interpreted as finite terms, *inductive theorem proving* is the standard approach for this kind of questions. However, here we deal with streams, that can be seen as infinite terms, and for which we need *co-induction* rather than induction.

For expressing the versions of the co-induction principle as we consider them, we need a fresh *freeze* symbol fr of type $s \to d$. This freeze symbol is used to force that the co-induction hypothesis is only used on top level, and not on subterms. This idea originates from [6], in which square brackets are used as the notation for the freeze operator.

For a set E' of equations of sort s we write $\mathsf{fr}(E')$ for the set of equations $\mathsf{fr}(\ell) = \mathsf{fr}(r)$ for $\ell = r \in E'$. The following theorem can be seen as an instance of Theorem 2 in [13]; for completeness we give a full proof based on our stream semantics and independent of the more abstract and general setting of [13].

▶ **Theorem 3.1** (Basic circular co-induction). *Let E be a stream specification and E' a set of equations of sort s, both not containing the symbol fr, such that*

- $E \models \mathsf{hd}(\ell) = \mathsf{hd}(r)$ *for all* $\ell = r \in E'$, *and*
- $E \cup \mathsf{fr}(E') \models \mathsf{fr}(\mathsf{tl}(\ell)) = \mathsf{fr}(\mathsf{tl}(r))$ *for all* $\ell = r \in E'$.

Then $E \models E'$.

Proof. Take an arbitrary stream model satisfying E; we have to prove that for all $\ell = r \in E'$ we have $[\ell, \alpha] = [r, \alpha]$ for every α, that is, $[\ell, \alpha](n) = [r, \alpha](n)$ for every $n \geq 0$. We do this by induction on n.

For $n = 0$ take $\ell = r \in E'$. We use the property $E \models \mathsf{hd}(\ell) = \mathsf{hd}(r)$ from which we conclude $[\mathsf{hd}(\ell), \alpha)] = [\mathsf{hd}(r), \alpha]$. We obtain $[\ell, \alpha](0) = [\mathsf{hd}(\ell), \alpha)] = [\mathsf{hd}(r), \alpha] = [r, \alpha](0)$.

For $n > 0$ we assume $[\ell, \alpha](n-1) = [r, \alpha](n-1)$ for every α and every $\ell = r \in E'$ as the induction hypothesis. For every $\ell = r \in E'$ we have to prove $[\ell, \alpha](n) = [r, \alpha](n)$, again for every α. As fr does neither occur in E nor in $\ell = r$, both the assumption that the model satisfies E and the induction hypothesis are independent of the symbol fr, and we are still free to define $[\mathsf{fr}]$. Let us define $[\mathsf{fr}](s) = s(n-1)$ for every stream s. Using the induction hypothesis, then for every α we have

$$[\mathsf{fr}(\ell), \alpha] = [\ell, \alpha](n-1) = [r, \alpha](n-1) = [\mathsf{fr}(r), \alpha],$$

so our model satisfies $\mathsf{fr}(\ell) = \mathsf{fr}(r)$ for all $\ell = r \in E'$. So from $E \cup \mathsf{fr}(E') \models \mathsf{fr}(\mathsf{tl}(\ell)) = \mathsf{fr}(\mathsf{tl}(r))$ we conclude that $[\mathsf{fr}(\mathsf{tl}(\ell)), \alpha] = [\mathsf{fr}(\mathsf{tl}(r)), \alpha]$, for all $\ell = r \in E'$. We obtain

$$[\ell, \alpha](n) = [\mathsf{fr}(\mathsf{tl}(\ell)), \alpha] = [\mathsf{fr}(\mathsf{tl}(r)), \alpha] = [r, \alpha](n),$$

concluding the proof. ◀

The set $\mathsf{fr}(E')$ in the second requirement of Theorem 3.1 is called the *co-induction hypothesis*.

▶ **Example 3.2.** Let E consists of the two equations $c = 0 : c$ and $d = 0 : d$. Then $E \models c = d$ is proved using Theorem 3.1 as follows. Choose $E' = \{c = d\}$. Here and in the following in equational reasoning we will abbreviate $=_{E \cup \mathbf{E}_{ht}}$ and $=_{E \cup \mathbf{E}_{ht} \cup \mathsf{fr}(E')}$ to $=$.

The first requirement $E \models \mathsf{hd}(c) = \mathsf{hd}(d)$ follows from

$$\mathsf{hd}(c) = \mathsf{hd}(0 : c) = 0 = \mathsf{hd}(0 : d) = \mathsf{hd}(d).$$

The second requirement follows from

$$\mathsf{fr}(\mathsf{tl}(c)) = \mathsf{fr}(\mathsf{tl}(0:c)) = \mathsf{fr}(c) = \mathsf{fr}(d) = \mathsf{fr}(\mathsf{tl}(0:d)) = \mathsf{fr}(\mathsf{tl}(d)),$$

using the co-induction hypothesis $\mathsf{fr}(c) = \mathsf{fr}(d)$. So Theorem 3.1 implies $E \models c = d$.

The basic machinery of our approach consists of combining Theorem 2.7 and Theorem 3.1: for proving $E \models \ell = r$ first it is tried to prove $\ell =_{E \cup \mathbf{E}_{ht}} r$. If this succeeds we are done, otherwise Theorem 3.1 is tried for $E' = \{\ell = r\}$. For the proof obligations of Theorem 3.1 again first Theorem 2.7 is tried, as was successful in Example 3.2. Where this fails, the failing proof obligation is added to E' after removing the fr symbols, and using this extended E' Theorem 3.1 is tried. This may be repeated any number of times. In a typical scenario in this way a finite set E' is found for which Theorem 3.1 applies. In this way not only validity of $\ell = r$ is proved, but also of any rule in E'.

▶ **Example 3.3.** Let E consist of the equations

$$\begin{array}{rclcrcl} \mathsf{zeros} & = & 0 : \mathsf{zeros} & & \mathsf{alt} & = & 0 : 1 : \mathsf{alt} \\ \mathsf{ones} & = & 1 : \mathsf{ones} & \mathsf{zip}(x:\sigma,\tau) & = & & x : \mathsf{zip}(\tau,\sigma). \end{array}$$

Then $E \models \mathsf{alt} = \mathsf{zip}(\mathsf{zeros}, \mathsf{ones})$ is proved using Theorem 3.1 as follows. First choose $E' = \{\mathsf{alt} = \mathsf{zip}(\mathsf{zeros}, \mathsf{ones})\}$. The first requirement follows from

$$\mathsf{hd}(\mathsf{alt}) = \mathsf{hd}(0:1:\mathsf{alt}) = 0 = \mathsf{hd}(0:\mathsf{zip}(\mathsf{ones},\mathsf{zeros}) =$$

$$\mathsf{hd}(\mathsf{zip}(0:\mathsf{zeros},\mathsf{ones}) = \mathsf{hd}(\mathsf{zip}(\mathsf{zeros},\mathsf{ones}).$$

For the second requirement we need $\mathsf{fr}(\mathsf{tl}(\mathsf{alt})) = \mathsf{fr}(\mathsf{tl}(\mathsf{zip}(\mathsf{zeros},\mathsf{ones})))$, for which equational reasoning using the co-induction hypothesis fails. So we add the equation $\mathsf{tl}(\mathsf{alt}) = \mathsf{tl}(\mathsf{zip}(\mathsf{zeros},\mathsf{ones}))$ to E' and try again to apply Theorem 3.1. By this addition now we obtain the requirements for the original equation in E' for free, and only need to consider the new equation. Now equational reasoning yields $\mathsf{hd}(\mathsf{tl}(\mathsf{alt})) = 1 = \mathsf{hd}(\mathsf{tl}(\mathsf{zip}(\mathsf{zeros},\mathsf{ones})))$, and for the second requirement we use the co-induction hypothesis $\mathsf{fr}(\mathsf{alt}) = \mathsf{fr}(\mathsf{zip}(\mathsf{zeros},\mathsf{ones}))$ to obtain

$$\mathsf{fr}(\mathsf{tl}(\mathsf{tl}(\mathsf{alt}))) = \mathsf{fr}(\mathsf{alt}) = \mathsf{fr}(\mathsf{zip}(\mathsf{zeros},\mathsf{ones})) = \mathsf{fr}(\mathsf{tl}(\mathsf{tl}(\mathsf{zip}(\mathsf{zeros},\mathsf{ones})))),$$

concluding the proof.

▶ **Example 3.4.** Consider streams over the naturals, that is, Σ_d consists of a constant 0 and a unary symbol s representing successor. Let E consist of the equations

$$\begin{array}{rclcrcl} \mathsf{from}(x) & = & x : \mathsf{from}(s(x)) & & & & \\ \mathsf{from2}(x) & = & x : \mathsf{from2}(s(s(x))) & \quad \mathsf{zip}(x:\sigma,\tau) & = & & x : \mathsf{zip}(\tau,\sigma). \end{array}$$

We will prove $E \models \mathsf{from}(x) = \mathsf{zip}(\mathsf{from2}(x), \mathsf{from2}(s(x)))$ using Theorem 3.1. Choose $E' = \{\mathsf{from}(x) = \mathsf{zip}(\mathsf{from2}(x), \mathsf{from2}(s(x)))\}$. The first requirement follows from

$$\mathsf{hd}(\mathsf{from}(x)) = \mathsf{hd}(x:\mathsf{from}(s(x))) = x = \mathsf{hd}(x:\mathsf{zip}(\mathsf{from2}(s(x)),\mathsf{from2}(s(s(x))))) =$$

$$\mathsf{hd}(\mathsf{zip}(x:\mathsf{from2}(s(s(x))),\mathsf{from2}(s(x)))) = \mathsf{hd}(\mathsf{zip}(\mathsf{from2}(x),\mathsf{from2}(s(x)))).$$

The second requirement follows from

$$\mathsf{fr}(\mathsf{tl}(\mathsf{from}(x))) = \mathsf{fr}(\mathsf{tl}(x:\mathsf{from}(s(x)))) = \mathsf{fr}(\mathsf{from}(s(x))) =_{\mathsf{fr}(E')}$$

$$\mathsf{fr}(\mathsf{zip}(\mathsf{from2}(s(x)),\mathsf{from2}(s(s(x))))) = \mathsf{fr}(\mathsf{tl}(x:\mathsf{zip}(\mathsf{from2}(s(x)),\mathsf{from2}(s(s(x)))))) =$$

$$\mathsf{fr}(\mathsf{tl}(\mathsf{zip}(x:\mathsf{from2}(s(s(x))),\mathsf{from2}(s(x))))) = \mathsf{fr}(\mathsf{tl}(\mathsf{zip}(\mathsf{from2}(x),\mathsf{from2}(s(x))))),$$

by which the claim has been proved by Theorem 3.1. In particular, for $x = 0$ this yields that the stream $\mathsf{from}(0)$ of natural numbers is the zip of the stream $\mathsf{from2}(0)$ of even numbers and the stream $\mathsf{from2}(s(0))$ of odd numbers.

More complicated examples over the natural numbers typically will require a combination of circular co-induction and induction over the natural numbers.

4 Special contexts

Next we present a powerful generalization of Theorem 3.1: instead of only assuming $\mathsf{fr}(\ell) = \mathsf{fr}(r)$ we may also assume $\mathsf{fr}(C[\ell]) = \mathsf{fr}(C[r])$ for *special contexts* C, where roughly speaking for every n the n-th element of the stream represented by $C[s]$ only depends on the first n elements of the stream represented by s. This notion originates from [6, 10]. As conceptually our notion is the same, we keep the terminology *special context* as introduced there.

▶ **Definition 4.1.** A context C of sort s and the hole of sort s is called *special* with respect to a stream specification, if for every stream model, for every α, every $n \geq 0$, and every pair of terms t, t' of sort s the following holds:

If $[t,\alpha](i) = [t',\alpha](i)$ for all $i \leq n$, then $[C[t],\alpha](n) = [C[t'],\alpha](n)$.

For example, the empty context $\Box$ is special; as a consequence of Theorem 4.5 we will conclude that $\mathsf{zip}(\Box,\sigma)$ and $\mathsf{zip}(\sigma,\Box)$ and $\mathsf{inv}(\Box)$ are special contexts with respect to the stream specification from Example 2.2.

As the first element of $\mathsf{tl}(t)$ is the second element of t, we observe that the context $\mathsf{tl}(\Box)$ is not special. In Example 4.3 we will show that $\mathsf{even}(\Box)$ is not a special context.

The requirement of a context C to be special is that $[C[\cdot],\alpha]$ is a *causal function* on streams, that is, for every n the the first n elements of the output stream only depend on the first n elements of the input stream. Such causal functions have been studied co-algebraically, e.g., in [8].

Since by definition the empty context is a special context, Theorem 3.1 is a direct consequence of the following theorem, which can be seen as an instance of Theorem 3 in [10]. For completeness we give a full proof based on our stream semantics and our corresponding definition of special context.

▶ **Theorem 4.2** (Circular co-induction with special contexts)**.** *Let E be a stream specification and E' a set of equations of sort s, both not containing the symbol fr, such that*

- $E \models \mathsf{hd}(\ell) = \mathsf{hd}(r)$ *for all* $\ell = r \in E'$*, and*
- $E \cup \{\mathsf{fr}(C[\ell]) = \mathsf{fr}(C[r]) \mid C$ *is a special context,* $\ell = r \in E'\} \models \mathsf{fr}(\mathsf{tl}(\ell)) = \mathsf{fr}(\mathsf{tl}(r))$ *for all* $\ell = r \in E'$*.*

Then $E \models E'$*.*

Proof. Take an arbitrary stream model satisfying E; we have to prove that $[\ell,\alpha] = [r,\alpha]$ for every α and every $\ell = r \in E'$, that is, $[\ell,\alpha](n) = [r,\alpha](n)$ for every $n \geq 0$. We do this by induction on n.

For $n = 0$ we use the property $E \models \mathsf{hd}(\ell) = \mathsf{hd}(r)$ from which we conclude $[\mathsf{hd}(\ell),\alpha)] = [\mathsf{hd}(r),\alpha]$. We obtain $[\ell,\alpha](0) = [\mathsf{hd}(\ell),\alpha)] = [\mathsf{hd}(r),\alpha] = [r,\alpha](0)$.

For $n > 0$ we assume as the induction hypothesis $[\ell,\alpha](i) = [r,\alpha](i)$ for every α and every $i < n$, for all $\ell = r \in E'$, and we have to prove $[\ell,\alpha](n) = [r,\alpha](n)$, again for every α and

$\ell = r \in E'$. As fr does neither occur in E nor in E', both the assumption that the model satisfies E and the induction hypothesis are independent of the symbol fr, and we are still free to define $[\mathsf{fr}]$. Just like in the proof of Theorem 3.1 define $[\mathsf{fr}](s) = s(n-1)$ for every stream s. Let C be any special context. Using the induction hypothesis and the definition of special context, for every α and $\ell = r \in E'$ we obtain

$$[\mathsf{fr}(C[\ell], \alpha] = [C[\ell], \alpha](n-1) = [C[r], \alpha](n-1) = [\mathsf{fr}(C[l], \alpha],$$

so our model satisfies $\mathsf{fr}(C[\ell]) = \mathsf{fr}(C[r])$. As this holds for every special context and $\ell = r \in E'$, and the model still satisfies E, from the condition of the theorem we conclude that the model also satisfies $\mathsf{fr}(\mathsf{tl}(\ell)) = \mathsf{fr}(\mathsf{tl}(r))$ for any $\ell = r \in E'$. Hence we obtain

$$[\ell, \alpha](n) = [\mathsf{fr}(\mathsf{tl}(\ell)), \alpha] = [\mathsf{fr}(\mathsf{tl}(r)), \alpha] = [r, \alpha](n),$$

concluding the proof. ◀

▶ **Example 4.3.** We start by a negative example. Let E consists of the four equations

$$\mathsf{even}(x : \sigma) = x : \mathsf{odd}(\sigma), \quad \mathsf{odd}(x : \sigma) = \mathsf{even}(\sigma), \quad c = 0 : \mathsf{even}(c), \quad d = 0 : \mathsf{even}(d).$$

We now will show that $\mathsf{even}(\Box)$ is not a special context. Consider two streams: one only consisting of zeros, and the other starting by two zeros, and followed by only ones. Then both satisfy the equations for c and d, so we conclude $E \not\models c = d$.

Next assume that $\mathsf{even}(\Box)$ is a special context. The first condition of Theorem 4.2 for proving $E \models c = \mathsf{zeros}$ is easily checked:

$$\mathsf{hd}(c) = \mathsf{hd}(0 : \mathsf{even}(c)) = 0 = \mathsf{hd}(0 : \mathsf{even}(d)) = \mathsf{hd}(d).$$

Using the co-induction hypothesis $\mathsf{fr}(\mathsf{even}(c)) = \mathsf{fr}(\mathsf{even}(d))$ also the second condition holds:

$$\mathsf{fr}(\mathsf{tl}(c)) = \mathsf{fr}(\mathsf{tl}(0 : \mathsf{even}(c))) = \mathsf{fr}(\mathsf{even}(c)) = \mathsf{fr}(\mathsf{even}(d)) = \mathsf{fr}(\mathsf{tl}(0 : \mathsf{even}(d))) = \mathsf{fr}(\mathsf{tl}(d)).$$

So by Theorem 4.2 we would conclude $E \models c = d$, contradicting the assumption that $\mathsf{even}(\Box)$ is a special context.

Theorem 4.2 becomes useful if we have a syntactic criterion to check whether a particular context is special. A suitable criterion is *friendly nestingness* as was introduced in [3] for establishing productivity. The underlying idea is that when using equations from left to right, at each step at most one stream element is consumed and at least one element is produced. Although the underlying idea is the same, for our purpose we need less conditions, for instance, we do not require orthogonality. In other settings this idea is also called *guardedness*, e.g., in process algebra, where a recursive specification is called *guarded* if right-hand sides can be rewritten to a choice among terms all having a constructor on top, see e.g. [2], Section 5.5.

To avoid confusion with the more restricted notion of friendly nestingness, we prefer to call it guardedness. Where the rest of this paper allows arbitrary equations, this guardedness criterion for detecting special contexts is the only spot where we focus on the pure stream format as given in [3, 14, 15].

▶ **Definition 4.4.** A stream specification E over $\Sigma \cup \Sigma_d \cup \{:, \mathsf{hd}, \mathsf{tl}\}$ is called *guarded* if no symbol of sort d in Σ has an argument of sort s, and all equations $\ell = r$ of E of sort s satisfy

- the symbols hd and tl do not occur in ℓ and r,
- r is not a variable, and the root of r is $:$,

- ℓ is of the shape $f(u_1 \ldots, u_n, t_1 \ldots, t_m)$ such that either $t_i \in \mathbf{X}_s$ or $t_i = x : \sigma$ for some $x \in \mathbf{X}_d, \sigma \in \mathbf{X}_s$, for every $i = 1, \ldots, m$,

and E is *exhaustive*, that is, for every term $t = f(u_1 \ldots, u_n, t_1 \ldots, t_m)$ for which $u_i \in D$ for $i = 1, \ldots, n$ and for every $i = 1, \ldots, m$ the term t_i is of the shape $d : t'$ for $d \in D$, there is such an equation $\ell = r$ and a substitution ρ such that $t = \ell\rho$.

▶ **Theorem 4.5.** *Let a stream specification E over $\Sigma \cup \Sigma_d \cup \{:, \mathsf{hd}, \mathsf{tl}\}$ be given, and subsets $E' \subseteq E$ and $\Sigma' \subseteq \Sigma$ such that E' over $\Sigma' \cup \Sigma_d \cup \{:, \mathsf{hd}, \mathsf{tl}\}$ is a guarded stream specification. Then every context over $\Sigma' \cup \Sigma_d \cup \{:\} \cup \mathbf{X}_s \cup \mathbf{X}_d$ of sort s is special with respect to E.*

Proof. Fix a model for E, and an assignment α. We prove the theorem by proving the following claim by induction on n:

> **Claim:** If terms t, t' of sort s satisfy $[t, \alpha](i) = [t', \alpha](i)$ for all $i \leq n$, then $[C[t], \alpha](i) = [C[t'], \alpha](i)$ for all $i \leq n$, for every context C of sort s in which all symbols on the path from the root to the hole (of sort s) are in $\Sigma' \cup \{:\}$.

For $C = \Box$ being the empty context the claim is trivial. For a context in which the hole is deeper then the first level, the claim can be proved by repeatedly applying instances of the claim for contexts in which the hole is immediately below the root. So it remains to prove the claim for $C = f(u_1, \ldots, u_n, t_1, \ldots, t_m)$ in which the hole $\Box$ is one of the arguments $t_1, \ldots, t_m$, say $t_I = \Box$; it can not be in a data argument since we assumed that data symbols have no arguments of sort s. For proving $[C[t], \alpha](i) = [C[t'], \alpha](i)$ for $i \leq n$ it is obtained form the induction hypothesis for $i < n$, we only need to prove $[C[t], \alpha](n) = [C[t'], \alpha](n)$.

In case $f = :$ we have $C = u_1 : \Box$. For $n = 0$ we conclude $[C[t], \alpha](0) = [u_1, \alpha] = [C[t'], \alpha](0)$, and for $n > 0$ we conclude

$$[C[t], \alpha](n) = [t_1[t], \alpha](n-1) = [t_1[t'], \alpha](n-1) = [C[t'], \alpha](n)$$

by the induction hypothesis.

It remains to prove $[C[t], \alpha](n) = [C[t'], \alpha](n)$ for $C = f(u_1, \ldots, u_n, t_1, \ldots, t_m)$ for $f \in \Sigma'$ and $t_I = \Box$. For any term v we have

$$[C[v], \alpha] = [f(u_1, \ldots, u_n, t_1, \ldots, t_{I-1}, v, t_{I+1}, \ldots, t_m)]), \alpha].$$

Here every u_i may be replaced by the data element $[u_1, \alpha)]$, and every t_i may be replaced by $[t_i, \alpha](0) : \mathsf{tl}(t_i)), \alpha]$, since $[t_i, \alpha] = [[t_i, \alpha](0) : \mathsf{tl}(t_i), \alpha]$ by Lemma 2.5, and similar for v. Now the resulting term matches with ℓ for some equation $\ell = r$ in E', due to the definition of guardedness, where the root of r is $:$. Then $[C[v], \alpha] = [u : C'[\mathsf{tl}(v)]^*, \alpha]$ for some $u \in D$ and some context C' having zero or more holes, and every path from the root to a hole only contains symbols from $\Sigma' \cup \{:\}$. The '$*$' in $C'[\mathsf{tl}(v)]^*$ means that every hole is filled by $\mathsf{tl}(v)$. For $n = 0$ we obtain $[C[t], \alpha](0) = [u : \cdots, \alpha](0) = [C[t'], \alpha](0)$, for $n > 0$ we apply the induction hypothesis on C' and the two terms $\mathsf{tl}(t)$ and $\mathsf{tl}(t')$. The condition of the induction hypothesis for $n - 1$ is $[\mathsf{tl}(t), \alpha](i) = [\mathsf{tl}(t'), \alpha](i)$ for all $i \leq n - 1$, which follows from the assumption $[t, \alpha](i) = [t', \alpha](i)$ for all $i \leq n$. So by the induction hypothesis we conclude $C'[\mathsf{tl}(t)]^*, \alpha](n-1) = [C'[\mathsf{tl}(t')]^*, \alpha](n-1)$.

In case C' has 1 hole it is immediate, in case C' has k holes this is concluded by applying the induction hypothesis k times. We conclude

$$\begin{aligned}
[C[t], \alpha](n) &= [u : C'[\mathsf{tl}(t)]^*, \alpha](n) \\
&= [C'[\mathsf{tl}(t)]^*, \alpha](n-1) \\
&= [C'[\mathsf{tl}(t')]^*, \alpha](n-1) \\
&= [u : C'[\mathsf{tl}(t')]^*, \alpha](n) \\
&= [C[t'], \alpha](n),
\end{aligned}$$

concluding the proof. ◀

▶ **Example 4.6.** Let E consist of the two equations

$$\mathsf{zip}(x:\sigma,\tau) = x:\mathsf{zip}(\tau,\sigma),\quad \mathsf{ones} = 1:\mathsf{zip}(\mathsf{ones},\mathsf{ones}).$$

We want to prove that $E \models \mathsf{ones} = 1:\mathsf{ones}$. Applying the standard approach based on Theorem 3.1 turns out to fail: E' will be extended forever by terms containing an increasing number of zip symbols. Instead, using Theorem 4.2 easily applies: according to Theorem 4.5 $\mathsf{zip}(\Box,\sigma)$ is a special context, by which we may use the co-induction hypothesis $\mathsf{fr}(\mathsf{zip}(\mathsf{ones},\sigma)) = \mathsf{fr}(\mathsf{zip}(1:\mathsf{ones},\sigma))$:

$$\begin{array}{lllll} \mathsf{fr}(\mathsf{tl}(\mathsf{ones})) & = & \mathsf{fr}(\mathsf{tl}(1:\mathsf{zip}(\mathsf{ones},\mathsf{ones}))) & = & \mathsf{fr}(\mathsf{zip}(\mathsf{ones},\mathsf{ones})) \\ & = & \mathsf{fr}(\mathsf{zip}(1:\mathsf{ones},\mathsf{ones})) & = & \mathsf{fr}(1:\mathsf{zip}(\mathsf{ones},\mathsf{ones})) \\ & = & \mathsf{fr}(\mathsf{ones}) & = & \mathsf{fr}(\mathsf{tl}(1:\mathsf{ones})). \end{array}$$

For proving $E \models \mathsf{ones} = 1:\mathsf{ones}$ by Theorem 4.2 it remains to prove $\mathsf{hd}(\mathsf{ones}) = \mathsf{hd}(1:\mathsf{ones})$, which is straightforward.

5 Exploiting unicity

Several techniques have been developed to prove that a stream specification has a unique solution, many of which are based on the notion of *productivity*. These techniques can be used for proving equality: if we want to prove $E \models c = t$ for some stream constant c having a unique solution, then we can try to prove that t satisfies the equations for c, that is, $E \models \ell\downarrow = r\downarrow$ for every equation $\ell = r$ in E containing the symbol c, where $\downarrow$ means that every symbol c is replaced by t. If this is the case, then both $[c]$ and $[t]$ satisfy the equations for c in every model, and since this is unique we conclude $[c] = [t]$ in every model, so proving $E \models c = t$. In this section we describe this approach in detail, also for functions rather than only for constants c, and give an example for which all earlier techniques fail and we make use of recent techniques to prove productivity.

▶ **Definition 5.1.** A function symbol $f \in \Sigma$ is called *uniquely defined* with respect to a set E of equations if $[f]_1 = [f]_2$ for every two models $[.]_1$, $[.]_2$ satisfying E.

▶ **Theorem 5.2.** *Let E be a set of equations such that a symbol f is uniquely defined with respect to E. Let $t = t'$ be an equation for which f is the root of t, and all arguments of f in t are variables, all distinct. Moreover, all variables in t' occur in t, and the symbol f does not occur in t'. Write $\downarrow$ for the normal formal with respect to the single rule $t \to t'$. Assume $E \models \ell\downarrow = r\downarrow$ for all equations $\ell = r$ in E. Then $E \models t = t'$.*

Proof. Let $[.]_1$ be any model satisfying E; we have to prove that $[t,\alpha]_1 = [t',\alpha]_1$ for every variable assignment α. Define $[.]_2$ by $[g]_2 = [g]_1$ for every symbol $g \neq f$, and let $[f]_2$ interpret t', that is, if $t = f(x_1,\ldots,x_n,\sigma_1,\ldots,\sigma_m$, then $[f]_2(d_1,\ldots,d_n,s_1,\ldots,s_m) = [t',\alpha]_1$ for α defined by $\alpha(x_i) = d_i$, $\alpha(\sigma_i) = s_i$. So by definition we have $[t,\alpha]_2 = [t',\alpha]_1$ for every α. Using this, one proves by induction on the structure of v that $[v,\alpha]_2 = [v\downarrow,\alpha]_1$ for every term v and every α. Hence for every equation $\ell = r$ in E and every α we obtain

$$[\ell,\alpha]_2 = [\ell\downarrow,\alpha]_1 = [r\downarrow,\alpha]_1 = [r,\alpha]_2,$$

so the model $[.]_2$ satisfies E. Since f is uniquely defined we now conclude $[f]_1 = [f]_2$, so

$$[t,\alpha]_1 = [t,\alpha]_2 = [t',\alpha]_1$$

for every α, concluding the proof. ◀

▶ **Example 5.3.** Let E consist of the equations

$$\begin{array}{rclcrcl} a & = & 0 : f(a) & & f(0:\sigma) & = & 1 : 0 : f(\sigma) \\ \mathsf{alt} & = & 0 : 1 : \mathsf{alt} & & f(1:\sigma) & = & f(\sigma). \end{array}$$

We want to prove that $E \models a = \mathsf{alt}$. Earlier techniques fail for doing this automatically, but we can apply Theorem 5.2 for the term $t = a$. First we have to prove that a is uniquely defined. This follows from productivity of all ground terms, which is proved using Theorem 4.1 from [15]. To this end context-sensitive termination of E has to be proved for the instance of context-sensitive rewriting in which rewriting is allowed on all arguments of all symbols except for the second argument of ":". This is easily proved fully automatically by tools like AProVE [4] and μ-Term [11].

Then by Theorem 5.2 it remains to prove that $E \models \ell\downarrow = r\downarrow$ for all equations $\ell = r$ in E, where $\downarrow$ means replacement of a by alt. As there is only one equation containing the symbol a, we only have to prove $E \models \mathsf{alt} = 0 : f(\mathsf{alt})$. This is easily proved by our basic machinery based on Theorem 3.1; it can also be proved by CIRC.

It is possible to prove $E \models a = \mathsf{alt}$ directly by Theorem 3.1 by choosing

$$E' = \{a = \mathsf{alt}, f^n(a) = 1 : \mathsf{alt}, 0 : f^n(a) = \mathsf{alt} \mid n > 0\}.$$

However, since E' is infinite, and the argument requires to prove $f^n(a) = 1 : 0 : f^{n+1}(a)$ for every $n > 0$, this approach is not suitable for automation.

We want to stress here that productivity of all ground terms does not imply that all functions are uniquely defined on all streams, only on stream interpretations of ground terms. In this example $[f]$ is not uniquely defined for all streams: one can make two distinct models both satisfying E in which $[f](s)$ are distinct, for s being the stream purely consisting of ones.

6 Implementation

We have implemented the techniques as presented in this paper in a tool called 'Streambox'. More precisely, any stream specification with a corresponding goal can be entered, or chosen from a list, and then the tool tries to prove the goal. The basic machinery is equational reasoning in combination with circular co-induction. For equational reasoning the tool simply searches for a conversion, so does not require termination. The tool Streambox is available on-line, as well as for download:

`http://infinity.few.vu.nl/streambox/`

The tool Streambox automatically proves equality of all examples included in this paper.

In this section we describe some aspects of our implementation not yet covered by the theory presented so far, in particular the implementation of *special contexts*, the use of *case analysis*, and *automatic lemma search*. Streambox is not stand alone: for exploiting unicity by using Theorem 5.2 it is tried to prove unicity by proving productivity by proving context-sensitive termination as described in [15] This is done by calling μ-Term [11], and in case this fails by calling AProVE [4]. In this way the power of state-of-the art termination provers is exploited.

6.1 Special contexts

Theorem 4.2 extends circular co-induction with special contexts. Roughly speaking, the idea is that to derive the equality $\mathsf{fr}(\mathsf{tl}(\ell)) = \mathsf{fr}(\mathsf{tl}(r))$, we are allowed to use equations of the form $\mathsf{fr}(C[\ell]) = \mathsf{fr}(C[r])$ for every special context C, in particular for contexts using symbols for which the specification is guarded, as discussed in Theorem 4.5. For example, the special contexts for Example 2.2 include:

$$\mathsf{inv}(\Box),\quad \mathsf{inv}(\mathsf{inv}(\Box)),\quad \mathsf{inv}(\mathsf{inv}(\mathsf{inv}(\Box))),\dots$$
$$\mathsf{zip}(\Box, s),\quad \mathsf{zip}(\mathsf{inv}(\Box), s),\quad \mathsf{zip}(\mathsf{inv}(\mathsf{inv}\Box), s),\quad \mathsf{zip}(\mathsf{zip}(\Box, s), t),\dots$$

Mimicking the use of these infinitely many special contexts can be done by moving the symbol fr up and down, as is described in the following lemma of which the proof is straightforward:

▶ **Lemma 6.1.** *Let E be a stream specification and E' a set of equations of sort s, both not containing the symbol* fr.

Let $\Sigma' \subseteq \Sigma$ such that every context over $\Sigma' \cup \{:\} \cup \mathbf{X}_s \cup \mathbf{X}_d$ of sort s is special. For every symbol $f \in \Sigma'$, let $f_\#$ be a fresh symbol of the same type. We define the set $\mathcal{S}(\Sigma')$ to consist of the following equations:

$$\mathsf{fr}(f(x_1,\dots,x_m,\sigma_1,\dots,\sigma_m)) = f_\#(x_1,\dots,x_m,\mathsf{fr}(\sigma_1),\dots,\mathsf{fr}(\sigma_m))$$

for every $f \in \Sigma'$.

Assume that the following conditions hold:

- *$E \models \mathsf{hd}(\ell) = \mathsf{hd}(r)$ for all $\ell = r \in E'$, and*
- *$E \cup \{\mathsf{fr}(\ell) = \mathsf{fr}(r)\} \cup \mathcal{S}(\Sigma') \models \mathsf{fr}(\mathsf{tl}(\ell)) = \mathsf{fr}(\mathsf{tl}(r))$ for all $\ell = r \in E'$.*

Then $E \models E'$. ◀

Streambox derives the set Σ' according to Theorem 4.5, and then employs the rules given in Lemma 6.1 to treat special contexts.

6.2 Case analysis

Let us consider the following specification:

$$\mathsf{inv}(0:\sigma) \;=\; 1:\mathsf{inv}(\sigma),\quad \mathsf{inv}(1:\sigma) \;=\; 0:\mathsf{inv}(\sigma),$$

and prove the equation $\mathsf{inv}(\mathsf{inv}(\sigma)) = \sigma$. A direct application of the basic circular co-induction principle fails: it needs the observation that every boolean stream σ is either of the shape $\sigma = 0:\sigma'$ or $\sigma = 1:\sigma'$.

The following straightforward lemma exploits this case analysis:

▶ **Lemma 6.2** (Case analysis)**.** *Let E be a stream specification such that* 0 *and* 1 *are the only data constructors. Let $E' \cup \{\ell = r\}$ be a set of equations, and σ a stream variable occurring in $\ell = r$. Then $E \models E' \cup \{\ell = r\}$ if and only if*

$$E \;\models\; E' \cup \{\ell[\sigma \mapsto 0:\sigma] = r[\sigma \mapsto 0:\sigma]\} \cup \{\ell[\sigma \mapsto 1:\sigma] = r[\sigma \mapsto 1:\sigma]\}$$

Applied to the above specification, the proof obligation is

$$\begin{aligned} E \cup \{\mathsf{fr}(\mathsf{inv}(\mathsf{inv}(\sigma))) = \mathsf{fr}(\sigma)\} \;\models\; & \{\mathsf{fr}(\mathsf{tl}(\mathsf{inv}(\mathsf{inv}(0:\sigma)))) = \mathsf{fr}(\mathsf{tl}(0:\sigma))\} \cup \\ & \{\mathsf{fr}(\mathsf{tl}(\mathsf{inv}(\mathsf{inv}(1:\sigma)))) = \mathsf{fr}(\mathsf{tl}(1:\sigma))\} \end{aligned}$$

which is easily proved by equational reasoning.

In Streambox the focus is on boolean streams, and only this boolean version of case analysis has been implemented. For other data structures case analysis can be employed as well. For example, for streams over natural numbers one can distinguish the cases $0 : \sigma$ and $\mathsf{s}(x) : \sigma$.

6.3 Automatic lemma search

For many examples, circular co-induction does not suffice for deriving the goal equations directly from the input system. Then it frequently helps to first find some auxiliary lemmas, which can themselves be proved using circular co-induction. These lemmas then may be employed to prove further lemmas, or the goal equations:

▶ **Lemma 6.3** (Lemma usage). *Let E be a stream specification and E', E'' sets of equations. If $E \models E'$, then $E \models E''$ if and only if $E \cup E' \models E''$.*

Proof. Since $E \models E'$ it follows that E' is valid in every stream model where E is valid. Hence, the set of stream models of $E \cup E'$ coincides with that of E. ◀

Lemma 6.3 is used as follows: if earlier techniques fail to prove $E \models E''$, then a set E' of small equations (the lemmas) is tried to be created for which $E \models E'$ can be proved, again using our approach, see below for more details. After every extension of E' by a new equation, it is tried to prove $E \cup E' \models E''$, and as soon this succeeds we are done. In proving new lemmas in E', it is allowed also to use earlier lemmas in E'.

The advantage with respect to circular co-induction is that the lemmas E' enrich the conversion relation, and thereby can be fruitful for deriving the goal equations.

Our tool Streambox supports an automated search for lemmas in the following way. It enumerates small terms $t_1, t_2, \ldots$ of sort s by increasing 'weight'. For the *weight* of a term t we have chosen the number of function symbols in t minus ξ-times the number of distinct variables in t (where $0 < \xi \leq 0.1$). The intention of this weight function is that the most general lemmas are encountered first. For example, it guarantees that $\mathsf{even}(\mathsf{zip}(s,t))$ is generated before before $\mathsf{even}(\mathsf{zip}(s,s))$, which in turn is found before $\mathsf{even}(\mathsf{zip}(\mathsf{ones},\mathsf{ones}))$.

When t_i is generated, for $j < i$ it is checked whether prefixes upon some depth are equal for t_i and t_j for replacing variables by some random streams. If this is the case, then it is tried to prove $E \models t_i = t_j$. If this succeeds, then $t_i = t_j$ is added to the set E' of lemmas.

▶ **Example 6.4.** Consider the following stream specification:

$$\begin{aligned}
\mathsf{morse} &= 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{morse}), \mathsf{tl}(\mathsf{morse})) & \mathsf{tl}(x : \sigma) &= \sigma \\
\mathsf{inv}(x : \sigma) &= \mathsf{not}(x) : \mathsf{inv}(\sigma) & \mathsf{not}(0) &= 1 \\
\mathsf{zip}(x : \sigma, \tau) &= x : \mathsf{zip}(\tau, \sigma) & \mathsf{not}(1) &= 0 \\
\mathsf{toeplitz} &= 1 : \mathsf{zip}(\mathsf{inv}(\mathsf{toeplitz}), \mathsf{ones}) & \mathsf{xor}(0 : x : xs) &= x : \mathsf{xor}(x : xs) \\
\mathsf{ones} &= 1 : \mathsf{ones} & \mathsf{xor}(1 : x : xs) &= \mathsf{not}(x) : \mathsf{xor}(x : xs)
\end{aligned}$$

with the goal of proving $\mathsf{toeplitz} = \mathsf{xor}(\mathsf{morse})$. Here morse is the Thue-Morse stream from Example 2.2, and $\mathsf{toeplitz}$ is a simple instance of a Toeplitz word as presented in [1], 10.11, exercise 42: $\mathsf{toeplitz}$ is the stream obtained by replacing the ?-symbols in the stream $101?^\omega$ consecutively by the elements of $\mathsf{toeplitz}$. The equality between $\mathsf{toeplitz}$ and this alternative characterization can also be proved by Streambox.

Our tool Streambox succeeds in proving $\mathsf{toeplitz} = \mathsf{xor}(\mathsf{morse})$ within approximately one minute. Among others, it discovers the 8 lemmas displayed below.

	Lemma	Uses Lemma
1	$\mathsf{xor}(\mathsf{tl}(s)) = \mathsf{tl}(\mathsf{xor}(s))$	
2	$\mathsf{xor}(\mathsf{inv}(s)) = \mathsf{xor}(s)$	1
3	$\mathsf{inv}(\mathsf{inv}(s)) = s$	
4	$\mathsf{inv}(\mathsf{ones}) = \mathsf{xor}(\mathsf{ones})$	1
5	$\mathsf{zip}(\mathsf{inv}(t), \mathsf{inv}(s)) = \mathsf{inv}(\mathsf{zip}(t, s))$	
6	$\mathsf{xor}(\mathsf{xor}(\mathsf{zip}(t, s))) = \mathsf{zip}(\mathsf{xor}(t), \mathsf{xor}(s))$	1
7	$\mathsf{xor}(\mathsf{zip}(\mathsf{inv}(t), s)) = \mathsf{inv}(\mathsf{xor}(\mathsf{zip}(t, s)))$	1, 2, 3, 5
8	$\mathsf{tl}(\mathsf{inv}(\mathsf{zip}(s, s))) = \mathsf{xor}(\mathsf{zip}(s, \mathsf{ones}))$	1
9	$\mathsf{toeplitz} = \mathsf{xor}(\mathsf{morse})$	1,2,3,4,5,6,7,8

Using these 8 lemmas, finally the proof is given.

In the default setting only the lemmas are shown that are really used, and all proofs are given in full detail, showing the use of lemmas and co-induction hypotheses in separate colors for clarity. By choosing 'options' in the tool several parameters can be changed.

7 Conclusions

Streams or infinite sequences have been study extensively, see e.g., [1] and its bibliography of over 70 pages. A compact way to specify streams is by giving a set of equations. In this paper we investigated techniques for proving equality of streams automatically, given by such sets of equations. As main achievements we summarize:

- We present self-contained theory for the circular co-induction principle for streams, including special contexts, as presented in [12, 5, 6, 9, 10, 13, 7]. Our proofs use the standard semantics of streams being maps from naturals to data.
- We allow any set of equations on streams; in particular we do not restrict to the specific format of behavioral equations as in [9, 10, 13] in which only equations are allowed with left hand sides having hd or tl on top. In contrast to the approach of the tool CIRC [9] we never require termination. Our default format is the pure stream format as used in [3, 14, 15], in which specifications are often given much shorter than by behavioral equations.
- Where in [10] it is stated that the algorithm for computing special contexts is quite complex, and hence not described in detail, we have a very simple criterion for checking for a powerful class of special contexts: *guarded contexts*, inspired by and closely related to *friendly nestingness* from [3].
- We present a technique to prove stream equality by exploiting unicity: two streams are equal if one of them is uniquely defined and the other satisfies the equations of the first. For using this we need to prove unicity automatically, for which we use a technique to prove productivity by means of context-sensitive termination.
- We offer a tool Streambox combining and exploiting these techniques fully automatically. In particular, for hard examples the tool searches for lemmas autonomously, and uses derived lemmas when appropriate. In this way for several stream equalities Streambox outperforms the earlier tool CIRC [9].

References

1 J.-P. Allouche and J. Shallit. *Automatic Sequences: Theory, Applications, Generalizations.* Cambridge University Press, 2003.

2 J. C. M. Baeten, T. Basten, and M. A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*, volume 50 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, Cambridge, UK, 2009.

3 J. Endrullis, C. Grabmayer, and D. Hendriks. Data-oblivious stream productivity. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08)*, volume 5330 of *Lecture Notes in Computer Science*, pages 79–96. Springer-Verlag, 2008. Web interface tool: `http://infinity.few.vu.nl/productivity/`.

4 J. Giesl et al. AProVE. Web interface and download: `http://aprove.informatik.rwth-aachen.de`.

5 J. Goguen, K. Lin, and G. Roşu. Circular coinductive rewriting. In *Proceedings, 15th International Conference on Automated Software Engineering (ASE'00).* Institute of Electrical and Electronics Engineers Computer Society, 2000. Grenoble, France, 11-15 September 2000.

6 J. Goguen, K. Lin, and G. Roşu. Conditional circular coinductive rewriting with case analysis. In *Recent Trends in Algebraic Development Techniques (WADT02)*, volume 2755 of *Lecture Notes in Computer Science*, pages 216–232. Springer, 2003.

7 E.-I. Goriac, D. Lucanu, and G. Roşu. Automating coinduction with case analysis. In *Twelfth International Conference on Formal Engineering Methods (ICFEM'10)*, volume 6447 of *Lecture Notes in Computer Science*, pages 220–236. Springer, 2010.

8 J. Kim. Coinductive properties of causal maps. In *Proceedings of the 12th International Conference on Algebraic Methodology and Software Technology (AMAST 2008)*, volume 5140 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2008.

9 D. Lucanu and G. Roşu. CIRC: A circular coinductive prover. In *CALCO'07*, volume 4624 of *Lecture Notes in Computer Science*, pages 372 – 378. Springer, 2007.

10 D. Lucanu and G. Roşu. Circular coinduction with special contexts. In *Proceedings of the 11th International Conference on Formal Engineering Methods (ICFEM'09)*, volume 5885 of *Lecture Notes in Computer Science*, pages 639–659. Springer, 2009.

11 S. Lucas et al. μ-Term. Web interface and download: `http://zenon.dsic.upv.es/muterm/`.

12 G. Roşu. *Hidden Logic.* PhD thesis, University of California at San Diego, 2000.

13 G. Roşu and D. Lucanu. Circular coinduction: A proof theoretical foundation. In *Proceedings of the 3rd International Conference on Algebra and Coalgebra in Computer Science (CALCO'09)*, volume 5728 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 2009.

14 H. Zantema. Well-definedness of streams by termination. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA'09)*, volume 5595 of *Lecture Notes in Computer Science*, pages 164–178. Springer-Verlag, 2009.

15 H. Zantema and M. Raffelsieper. Proving productivity in infinite data structures. In Christopher Lynch, editor, *Proceedings of the 21st International Conference on Rewriting Techniques and Applications*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 401–416, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.