Equality of Streams is a Π^0_2-Complete Problem

Grigore Roşu
Department of Computer Science, University of Illinois at Urbana-Champaign
grosu@cs.uiuc.edu

Abstract

This paper gives a precise characterization for the complexity of the problem of proving equal two streams defined with a finite number of equations: Π^0_2. Since the Π^0_2 class includes properly both the recursively enumerable and the co-recursively enumerable classes, this result implies that one can find no mechanical procedure to say when two streams are equal, as well as no procedure to say when two streams are not equal. In particular, there is no complete proof system for equality of streams and no complete system for dis-equality of streams.

1. Introduction

Streams can be equivalently regarded as functions on natural numbers in a trivial way, by associating to each natural number n the element on the n-th position in the stream. Since the equality of functions on natural numbers is an arbitrarily complex problem, so is the equality of streams in general. However, there is a relatively broad interest in streams defined in a particular but intuitive and meaningful way, namely equationally. For example, the usual *zeros* and *ones* streams containing only 0 and 1 bits, respectively, as well as a *blink* stream of alternating 0 and 1 bits and the *zip* binary operation on streams, can be defined equationally as follows:

$$\begin{aligned} &\mathit{zeros} = 0 : \mathit{zeros} \\ &\mathit{ones} = 1 : \mathit{ones} \\ &\mathit{blink} = 0 : 1 : \mathit{blink} \\ &\mathit{zip}(B : S, S') = B : \mathit{zip}(S', S) \end{aligned}$$

Lazy evaluation languages, such as Haskell, support streams defined equationally as above.

Streams can be formally defined many different, but ultimately equivalent ways; e.g., as a coinductive type [4], as a final coalgebra [8], as an observational specification [2] or as a hidden logic theory [6]. All these approaches build upon the observation that streams cannot be defined using ordinary algebraic arguments, such as the ordinary semantics of equational specifications; one reason for this is that equational specifications allow too many models, making it impossible to prove any meaningful property on streams. Consider, for example, infinite streams of bits together with their usual constructor $\_ : \_$ and together with the streams and stream operations defined equationally above. Then note that the expected properties $\mathit{zip}(\mathit{zeros}, \mathit{zeros}) = \mathit{zeros}$, $\mathit{zip}(\mathit{ones}, \mathit{ones}) = \mathit{ones}$ and $\mathit{zip}(\mathit{zeros}, \mathit{ones}) = \mathit{blink}$ cannot be proved equationally, not even making use of induction, because they actually do *not* hold in the initial model of the equations above. Indeed, in the initial model, *zeros* and $\mathit{zip}(\mathit{zeros}, \mathit{zeros})$ are two different equivalence classes of terms, both closed under concatenation with 0; there is nothing in the ordinary equational setting to make such stream terms equal.

There are several approaches to proving streams equal, such as *coinduction* in a coalgebraic equational specification of streams [8], *context induction* [5] in an observational equational framework, or *circular coinduction* [6] in a hidden logic framework; the first two need human support, while the latter is automatic. By circular coinduction, for example, one can prove the equality $\mathit{zip}(\mathit{zeros}, \mathit{ones}) = \mathit{blink}$ as follows: (1) check that the two streams have the same head, 0; (2) take the tail of the two streams and generate the corresponding new goal $\mathit{zip}(\mathit{ones}, \mathit{zeros}) = \mathit{1{:}blink}$, which becomes the next task; (3) check that the two new streams have the same head, 1; (4) take the tail of the two new streams; after simplification one gets the new goal $\mathit{zip}(\mathit{zeros}, \mathit{ones}) = \mathit{blink}$, which is nothing but the original proof task; (5) conclude that $\mathit{zip}(\mathit{zeros}, \mathit{ones}) = \mathit{blink}$ holds. The intuition for the above "proof" is that the two streams have been exhaustively tried to be distinguished by iteratively checking their heads and taking their tails. Ending up in circles (we obtained the same new proof task as the original one) means that the two streams are indistinguishable, so equal.

Since some algorithms and/or proof systems can show many challenging equalities of streams and seem not to fail even on large and tricky examples, one may be (wrongly) tempted to prove them complete; by a complete algorithm in this context we mean one which answers "yes" on precisely the inputs consisting of pairs of equal streams - on the others it may either not terminate, or terminate with an output different from "yes". Also, since equational logic is complete, one may (wrongly) think that streams defined equationally must also admit some complete proof system. Moreover, since two different streams must differ on some position of finite index, one could (also wrongly) think that at least one can detect when two streams are not equal. The Π^0_2-completeness result in this paper tells us that there is actually *no* algorithm or proof system that is complete for equality of streams in general, as well as *no* algorithm or proof system that is complete for dis-equality of streams. Recall that Π^0_2 is the class in the arithmetic hierarchy which properly extends both classes r.e. (recursively enumerable) and co-r.e., and contains predicates of the form $P(a) := (\forall x)(\exists y)R(a, x, y)$ where R is a primitive recursive predicate [7].

Despite its pessimistic flavor, this impossibility result actually tells us two important facts: (1) that we should focus our efforts on exploring heuristics or deduction rules to prove or disprove equalities of streams that *work well on examples of interest* rather than in general, and (2) that further restrictions are needed on the original equational definition of the two streams in order to have complete deduction systems or algorithms. Note, however, that the equational definition that we will use to show the Π^0_2-hardness is very basic: it contains a finite set of binary predicates on streams,

which can be actually replaced by binary predicates involving no streams but only finite terms, together with only one operation producing a stream that is defined in a guarded style, similarly to the definition of *zip*. Therefore, (2) is probably hard to achieve satisfactorily.

This paper is structured as follows. Section 2 defines streams formally and shows that the satisfaction problem of an equality of streams belongs to the class Π^0_2. Section 3 shows how Turing machines can be specified both equationally and in terms of streams, and how computation in a Turing machine can be regarded as both equational deduction and rewriting. Finally, Section 4 shows how to reduce the equality problem of streams to a problem known to be Π^0_2-complete, thus proving the remaining Π^0_2-hardness result. Finally, Section 5 concludes the paper.

2. Streams and Membership to Π^0_2

Streams and equational definitions of streams can be formalized in many different, but ultimately equivalent ways. In this paper we choose to consider streams defined behaviorally, because this approach appears to require a minimal amount of formalization. Behavioral equational theories differ from ordinary equational theories in that equality is *behavioral* with respect to certain operations, called behavioral or observational, in the sense of indistinguishability under experiments performed with those operations.

Streams have been shown to be definable as coalgebras [8], as observational specifications [2], and as behavioral, or hidden specifications [6], with only two observers for experiments, namely taking the head and the tail of a stream; also, we have shown that the satisfaction problem for behavioral specifications over finite data in general is Π^0_2-complete [3]. From these, we can deduce that the stream equality problem belongs to the class Π^0_2.

In this paper we give a precise characterization of the stream equality problem, namely Π^0_2-*complete*, thus showing that streams, this canonical example of coalgebra, observational and/or behavioral specification, already carries the worst-case complexity of behavioral equivalence. The proof of Π^0_2-completeness of behavioral satisfaction in behavioral logics presented in [3] was *not* based on streams: it was based on a more complex encoding of the same Π^0_2-complete problem considered here.

We prefer to translate the behavioral Π^0_2 membership argument to streams, thus making this paper self-contained and the result more insightful for streams, as opposed to proving that streams form a particular behavioral theory and then using the general Π^0_2 membership result. The reader is *not* assumed familiar with observational specification, behavioral satisfaction or coalgebra.

The membership of the stream equality problem to the Π^0_2 class follows by the completeness of first-order reasoning with equality: we show that for any head/tail experiment, i.e., any position in the two streams, there is some proof using standard first-order reasoning that the elements corresponding to that position in the two streams are equal. To make precise the connection between equational definitions of streams and the completeness of first-order reasoning, we take a semantic approach. From now on we fix two sorts *Bit* and *Stream*, constants 0 and 1 of sort *Bit*, an operation $\_ : \_$ of arity *Bit* $\times$ *Stream* $\rightarrow$ *Stream*, two operations *head* : *Stream* $\rightarrow$ *Bit* and *tail* : *Stream* $\rightarrow$ *Stream* that we also call *behavioral* or *observers* or *deconstructors*, and two equations

$$\begin{aligned} &\mathit{head}(B:S) = B \\ &\mathit{tail}(B:S) = S. \end{aligned}$$

We let capital letters denote variables and do not mention their sorts when they can be inferred from the context; for example, in the equations above the sort of B is *Bit* and that of S is *Stream*. Equations are assumed quantified universally by the variables that appear in them. An *equational definition of streams* is an equational theory, say $\mathcal{S}$, consisting of the above signature possibly extended with other operations, and of the above equations possibly extended with other equations over the extended signature. In this paper it suffices to consider only *finite* equational theories.

Let $\mathcal{S}$ be an equational definition of streams (Σ, E), where Σ is its extended signature and E is its (finite) set of equations including the two above. Note that Σ can have more sorts that just *Bit* and *Stream*; in this paper, we also show examples referring to a sort *List* for finite lists of bits.

Example. An equational definition of streams $\mathcal{S}$ can include, for example, all those classic operations and equations in Section 1, as well as the operations *odd* and *even*, also standard, that return the streams of elements on the odd and even positions in a given stream, respectively, defined mutually recursively as follows:

$$\begin{aligned} &\mathit{odd}(B:S) = B:\mathit{even}(S) \\ &\mathit{even}(B:S) = \mathit{odd}(S). \end{aligned}$$

Also, one can define a sort *List* for finite lists of bits together with a special constant list *nil* : $\rightarrow$ *List*, and overload the constructor operation $\_ : \_$ to one of arity *Bit* $\times$ *List* $\rightarrow$ *List*. Then one can define an operation *repeat* : *List* $\rightarrow$ *Stream* generating a stream repeating infinitely a finite list as follows:

$$\begin{aligned} &\mathit{repeat}(L) = \mathit{aux}(L, L) \\ &\mathit{aux}(B:L', L) = B:\mathit{aux}(L', L) \\ &\mathit{aux}(nil, L) = \mathit{aux}(L, L), \end{aligned}$$

where *aux* : *List* $\times$ *List* $\rightarrow$ *Stream* is an auxiliary operation.

Intuition and common sense tell us that the definitions above are all "correct", in the sense that they indeed define unique behaviors for the corresponding operations. But how about a stream m (or a constant operation m : $\rightarrow$ *Stream*) defined equationally as $m = 0 : \mathit{even}(m)$? This equation is not a correct definition of m, because it admits more than one solution: e.g., both the stream of zeros and the stream starting with zero and followed by ones. We do not investigate the interesting problem of well-definedness of streams here, but only mention that one possible definition of well-definedness can be given semantically: a stream is well-defined in a given equational specification $\mathcal{S}$ iff it has the same elements in the same order in any model of $\mathcal{S}$. □

A *model of streams* is any set A_{Stream} together with two functions $A_{\mathit{head}} : A_{\mathit{Stream}} \rightarrow \{0,1\}$ and $A_{\mathit{tail}} : A_{\mathit{Stream}} \rightarrow A_{\mathit{Stream}}$. Given any model of streams $(A_{\mathit{Stream}}, A_{\mathit{head}}, A_{\mathit{tail}})$, we can define a behavioral equivalence as *indistinguishability under experiments with head and tail* as follows: $a, a' \in A_{\mathit{Stream}}$ are *behaviorally equivalent*, written $a \equiv a'$, iff $A_\gamma(a) = A_\gamma(a')$ for any $\{\mathit{head}, \mathit{tail}\}$-experiment γ, i.e., a *head* followed by a finite number of *tail* operations, where $A_{\mathit{head}(\mathit{tail}(\cdots \mathit{tail}(*)))}(a)$ is a shorthand for $A_{\mathit{head}}(A_{\mathit{tail}}(\cdots A_{\mathit{tail}}(a)))$. In other words, two streams in the model A_{Stream} are behaviorally equivalent iff they can produce the same elements in the same order when requested. Note that some models can encode streams in less conventional ways, e.g., as infinite binary trees traversed in breadth-first order when requested to produce elements (with A_{head} and A_{tail}), or as real numbers, etc. It is easy to see that $\equiv$ is a congruence for A_{head} and A_{tail} (i.e., if $a \equiv a'$ then $A_{\mathit{head}}(a) = A_{\mathit{head}}(a')$ and $A_{\mathit{tail}}(a) \equiv A_{\mathit{tail}}(a')$), and that it is the *largest* such congruence. One can easily show that models of streams are particular *hidden algebras* in the sense of [6], or particular *coalgebras* over a functor *Set* $\rightarrow$ *Set* that takes a set X to $\{0,1\} \times X$ [8]. The reader needs not be familiar with observational logic, or hidden algebra or coalgebra; we just want to re-emphasize that streams can be formalized many (apparently) different but ultimately equivalent ways.

If Σ is a signature over streams, then a Σ-*model of streams*, or simply a Σ-*model*, is a Σ-algebra A that protects the bits and preserves the behavioral equivalence on streams, i.e., a pair con-

sisting of a family of sets (called *carriers*) $\{A_s \mid s \in \mathit{Sorts}(\Sigma)\}$, in particular a set of "streams" A_{Stream}, and of a family of functions $\{A_\sigma : A_{s_1} \times A_{s_n} \to A_s \mid \sigma : s_1 \times s_n \to s \in \mathit{Operations}(\Sigma)\}$, with the following important restrictions: $(A_{\mathit{Stream}}, A_{\mathit{head}}, A_{\mathit{tail}})$ is a model of streams and its behavioral equivalence $\equiv$ is a Σ-congruence, $A_{\mathit{Bit}} = \{0, 1\}$, $A_0 = 0$, and $A_1 = 1$. We ambiguously let $\equiv$ denote the behavioral equivalence on A_{Stream} extended with identity relations on all the other sorts, including A_{Bit}. Let $\mathit{str} = \mathit{str'}$ be a Σ-equation over variables V and A a Σ-model. Then A *(behaviorally) satisfies* the equation $(\forall V)\ \mathit{str} = \mathit{str'}$, written $A \models (\forall V)\ \mathit{str} = \mathit{str'}$, iff $\theta^*(\mathit{str}) \equiv \theta^*(\mathit{str'})$ for any (appropriate many-sorted) mapping $\theta : V \to A$, where θ^* is its unique extension to Σ-terms. If V is empty then, for aesthetic reasons, we omit it as part of an equation. If $\mathcal{E}$ is an equational specification (Σ, E), then we write $A \models \mathcal{E}$ for some Σ-model A whenever A behaviorally satisfies all the equations in E, and $\mathcal{E} \models e$ for some Σ-equation e whenever $A \models \mathcal{E}$ implies $A \models e$ for all Σ-models A. One can now easily show that the specification in Section 1 behaviorally satisfies equations like $\mathit{zip}(\mathit{zeros}, \mathit{ones}) = \mathit{blink}$. We next define the *stream equality problem* formally:

> INPUT: An equational stream definition $\mathcal{E} = (\Sigma, E)$ and a Σ-equation e of sort *Stream*;
> OUTPUT: $\mathcal{E} \models e$?

PROPOSITION 1. *If $\mathcal{E} = (\Sigma, E)$, A is a Σ-model of streams, and e is any Σ-equation, then*

(1) $A \models e$ iff $A/_\equiv \models e$, where the latter is standard first-order with equality ($\mathrm{FOL}_=$*) satisfaction;*
(2) If e has the sort Bit, then $\mathcal{E} \models e$ iff $\mathcal{E}_{\mathit{Bit}} \models e$, where the latter is standard satisfaction in $FOL_=$ and $\mathcal{E}_{\mathit{Bit}}$ is the $FOL_=$ specification extending $\mathcal{E}$ with the formulae $\neg(0 = 1)$ and $(\forall B{:}\mathit{Bit})\ B = 0 \vee B = 1$.

Proof: *(1)* If the sort of the equation e is different from *Stream*, then *(1)* holds vacuously because $\equiv$ is the identity relation in A on those sorts. Now suppose that e is an equation of sort *Stream* of the form $(\forall V)\ \mathit{str} = \mathit{str'}$. If $A \models e$ then let $\theta_\equiv : V \to A/_\equiv$ be any "valuation" mapping of V into $A/_\equiv$, and let $\theta : V \to A$ be a mapping "choosing" for each stream equivalence class $\theta_\equiv(v)$ in $A/_\equiv$ an arbitrary representative, $\theta(v)$. Since $A \models e$, it follows that $\theta^*(\mathit{str}) \equiv \theta^*(\mathit{str'})$. On the other hand, since $\equiv$ is a congruence for all the operations in Σ, it follows that the equivalence classes of $\theta^*(\mathit{str})$ and $\theta^*(\mathit{str'})$ are precisely $\theta^*_\equiv(\mathit{str})$ and $\theta^*_\equiv(\mathit{str'})$, respectively. Therefore, $\theta^*_\equiv(\mathit{str}) = \theta^*_\equiv(\mathit{str'})$, that is, $A/_\equiv \models e$. Conversely, if $A/_\equiv \models e$ then let $\theta : V \to A$ be any mapping and define $\theta_\equiv : V \to A/_\equiv$ where $\theta_\equiv(v)$ is the equivalence class of $\theta(v)$. As above, since $\equiv$ is a congruence for the operations in Σ, it follows that the equivalence classes of $\theta^*(\mathit{str})$ and $\theta^*(\mathit{str'})$ are precisely $\theta^*_\equiv(\mathit{str})$ and $\theta^*_\equiv(\mathit{str'})$, respectively. On the other hand, since $A/_\equiv \models e$, it follows that $\theta^*_\equiv(\mathit{str}) = \theta^*_\equiv(\mathit{str'})$, which implies that $\theta^*(\mathit{str}) \equiv \theta^*(\mathit{str'})$. Therefore, $A \models e$.

(2) Let e be an equation of sort *Bit*, and let us first assume that $\mathcal{E} \models e$. To show that $\mathcal{E}_{\mathit{Bit}} \models e$, let us pick some Σ-algebra A such that $A \models \mathcal{E}_{\mathit{Bit}}$. Since $\neg(0 = 1)$ and $(\forall B{:}\mathit{Bit})\ B = 0 \vee B = 1$ are in $\mathcal{E}_{\mathit{Bit}}$, it follows that the carrier A_{Bit} of A has only two elements and those correspond to the constant operations 0 and 1. Also, since $A \models \mathcal{E}$ and since $\equiv$ includes the identity relation, it follows also that $A \models \mathcal{E}$. Since $\mathcal{E} \models e$, it follows that $A \models e$. Now since e is of sort *Bit* on which the equivalence relation $\equiv$ is precisely the identity, it follows that $A \models e$. Therefore, $\mathcal{E}_{\mathit{Bit}} \models e$. Conversely, let $\mathcal{E}_{\mathit{Bit}} \models e$ and let A be any Σ-model of streams such that $A \models \mathcal{E}$. By *(1)*, it follows that $A/_\equiv \models \mathcal{E}$. Moreover, since A_{Bit} contains precisely two elements which correspond to the constants 0 and 1, and since $\equiv$ is the identity on the sort *Bit*, it follows that $A/_\equiv \models \mathcal{E}_{\mathit{Bit}}$, so $A/_\equiv \models e$. By *(1)* it now follows that $A \models e$. Therefore, $\mathcal{E} \models e$. □

THEOREM 1. ***(Π^0_2-membership)*** *For any equational stream definition $\mathcal{E} = (\Sigma, E)$ and any Σ-equation $(\forall V)\ \mathit{str} = \mathit{str'}$ of sort Stream, $\mathcal{E} \models (\forall V)\ \mathit{str} = \mathit{str'}$ if and only if $\mathcal{E}_{\mathit{Bit}} \models (\forall V)\ \gamma(\mathit{str}) = \gamma(\mathit{str'})$ in* $\mathrm{FOL}_=$ *for all $\{\mathit{head}, \mathit{tail}\}$-experiments γ, i.e., terms of the form head followed by a finite number of tail operations. In particular, the stream equality problem is Π^0_2.*

Proof: The first part follows by *(2)* in Proposition 1, noticing that $\mathcal{E} \models (\forall V)\ \mathit{str} = \mathit{str'}$ if and only if $\mathcal{E} \models (\forall V)\ \gamma(\mathit{str}) = \gamma(\mathit{str'})$ for all $\{\mathit{head}, \mathit{tail}\}$-experiments γ, the latter following immediately from the definition of $\models$. For the membership to the Π^0_2 class part we use the fact that $\models$ in $\mathrm{FOL}_=$ admits complete deduction: $\mathcal{E} \models (\forall V)\ \mathit{str} = \mathit{str'}$ if and only if (for any experiment γ, there is some equational proof π_γ such that $\mathcal{E}_{\mathit{Bit}} \vdash^{\pi_\gamma} (\forall V)\ \gamma(\mathit{str}) = \gamma(\mathit{tr'})$). Since checking a given first-order proof is a decidable problem, the problem of proving stream equality belongs to the class Π^0_2. □

Note that we have under-used the completeness of $\mathrm{FOL}_=$ in the proof of membership to Π^0_2 above, because the $\mathrm{FOL}_=$ specification $\mathcal{E}_{\mathit{Bit}}$ has only two non-equational formulae, namely the negation and the disjunction in *2* in Proposition 1. When proving membership to a complexity class, for the sake of generality, one would like to show it for as unrestricted problems as possible. In our case, we believe that the Π^0_2-membership result above holds for a larger class of stream definitions than just equational. However, as shown in [6], there are some intricate technical problems with conditional equations when conditions have a "hidden" sort, *Stream* in our case, related to the intuition that one needs an infinity number of experiments to check whether a conditional equation apply; in particular, a result like the one in Proposition 1 does not hold. This makes us conjecture that the Π^0_2-membership result can be extended to more general $\mathrm{FOL}_=$ definitions of streams $\mathcal{E}$ in which equations of streams appear only on *positive* positions, i.e., ones without negations preceding them. In this paper, however, we limit ourselves to just equational definitions of streams.

We will show in Section 4 that the equality problem of two equationally defined streams is actually Π^0_2-*complete*. The hardness part of the proof is by reduction to a problem known to be Π^0_2, namely the TOTALITY problem saying whether a Turing machine halts on all inputs, or, in other words, whether a partially recursive function is total. The Π^0_2-hardness results that follow work for equational definitions that are so basic that they would straightforwardly imply the Π^0_2-completeness of the stream equality problem for any stream definitional setting including equations of streams for which a property of the form $\mathcal{E} \models e$ iff $\mathcal{E}_{\mathit{Bit}} \models e$ (like the one in *2* in Proposition 1) holds for any equation of sort *Bit*, where the latter satisfaction relation admits complete deduction.

3. Encoding Computation by Equational Deduction and Rewriting

Equational encodings of general computation into equational deduction are well-known; for example, [1] shows such equational encodings of computation, where the resulting equational specifications, if regarded as term rewrite systems (TRSs), are confluent and terminate whenever the original computation terminates. Our goal in this section is to discuss equational encodings of (Turing machine) computation. These encodings will be used in Section 4 to show the Π^0_2 hardness of the stream equality problem. Since we believe that the Π^0_2-hardness result can be useful in other settings as well, for example in the context of infinitary rewriting, we pay special attention to the *minimality* of the subsequent encodings. By minimal encoding in this setting we mean *restrictive* resulting equational specifications. For example, an equational specification

which, when regarded as a TRS, is confluent and terminates on certain input terms is regarded as a restricted equational specification; also, an equational specification using only sorts *Bit* and *Stream* is regarded as restrictive, and so is considered the fact that the task to prove is a ground (with no variables) equality. The equational encodings that follow can be faithfully used as TRS Turing-complete computational engines, because each rewrite step corresponds to precisely one computation step in the Turing machine. We discuss two encodings in the sequel, the first suggesting the second. The first encoding allows the use of any rewrite engine, with no reduction strategies, as a computational engine, while the second requires the rewrite engine to use lazy evaluation.

3.1 Turing Machines

There are many equivalent definitions of Turing machines in the literature. We prefer one adapted from [7], and describe it informally in the sequel. The reader is assumed familiar with basics of Turing machines, the role of the following paragraphs being to establish our notations and conventions. Consider a mechanical device which has associated with it a tape of infinite length in both directions, partitioned in spaces of equal size, called *cells*, which are able to hold either a "0" or an "1" and are rewritable. The device examines exactly one cell at any time, and can perform any of the following four operations (or *commands*):

1. Write a "1" in the current cell;
2. Write a "0" in the current cell;
3. Shift one cell to the right;
4. Shift one cell to the left.

The device performs one operation per unit time, and this performance is called a *step*. Formally, let Q be a finite set of *internal states*, containing a *starting state* q_s and a *halting state* q_h. Let $B = \{0, 1\}$ be a set of *symbols* (or *bits*) and $C = \{0, 1, \rightarrow, \leftarrow\}$ be a set of *commands*. Then a (deterministic) Turing machine is a mapping M from $Q \times B$ to $Q \times C$. We assume that the tape contains only 0's (or blanks) before the machine starts performing. A *configuration* of a Turing machine is a triple consisting of an internal state and two infinite strings (notice that the two infinite strings contain only 0's starting with a certain cell), standing for the cells on the left and for the cells on the right, respectively. We let $(q, L|R)$ denote the configuration in which the machine is in state q, with left tape L and right tape R.

Given a configuration $(q, L|R)$, the content of the tape is LR, which is infinite at both ends. By convention, the current cell is the first cell of the right string. We also let $(q, L|R) \rightarrow (q', L'|R')$ denote the configuration transition under one of the four commands. Given a configuration in which the internal state is q and the examined cell contains b, and if $\delta(q, b) = (q', c)$, then exactly one of the following configuration transitions can take place:

1. $(q, L|bR) \rightarrow (q', L|cR)$ if $c = 0$ or $c = 1$;
2. $(q, L|bR) \rightarrow (q', Lb|R)$ if $c = \rightarrow$;
3. $(q, Lb'|bR) \rightarrow (q', L|b'bR)$ if $c = \leftarrow$.

The machine starts performing in the internal state q_s. If there is no input, the initial configuration on which the Turing machine is run is $(q_s, \cdots 0 \cdots 0|0 \cdots 0 \cdots)$. Sometimes, we wish to run a Turing machine on a specific input, say $x = b_1 b_2 \cdots b_n$. In this case, its initial configuration is $(q_s, \cdots 0 \cdots 0|b_1 b_2 \cdots b_n 0 \cdots 0 \cdots)$. A Turing machine *stops* when it first gets to its halting state, q_h. Therefore, a Turing machine carries out a uniquely determined succession of steps, which may or may not terminate. It is well-known that Turing machines can compute exactly the partial recursive functions [7]. From here on, let us fix a Turing machine M.

3.2 An Encoding Without Streams

Since at any moment the computation that already took place has only used a finite number of cells, we can simulate the infinite tape with two finite, but potentially arbitrarily long, tapes. Let us assume finite lists of bits. These can be defined algebraically with two sorts, say *Bit* and *List*, two constants 0 and 1 of sort *Bit*, one constant *nil* of sort *List*, and one constructor operation $\_ : \_$ of arity *Bit* $\times$ *List* $\rightarrow$ *List*. Let us also consider an operation q : *List* $\times$ *List* $\rightarrow$ *Bit* for each state $q \in Q$; these operations corresponding to states in the Turing machine can be regarded as "wrappers" of the infinite tape; the equational specification and its corresponding TRS below will encode the computation of M making use of terms of the form $q(L, R)$, regarded as configurations of M: the machine is in state q, with left tape L and right tape R. With this intuition, we can now naturally encode each transition $\delta(q, b) = (q', c)$ in M where $q \neq q_h$ with precisely one equation, as follows:

$$q(L, b : R) = q'(L, c : R) \text{ if } c \text{ is 0 or 1,}$$
$$q(L, b : R) = q'(b : L, R) \text{ if } c \text{ is } \rightarrow, \text{ or}$$
$$q(b' : L, b : R) = q'(L, b' : b : R) \text{ if } c \text{ is } \leftarrow.$$

To state that the computation ends when the state q_h is reached, we add the equation

$$q_h(L, R) = 1.$$

The equations above treat the common cases in which the tapes are not *nil* when their first elements are needed. As shown later, if one uses infinite streams instead of finite lists then the above equations are sufficient. In the context of finite lists, to complete the definition, we also need to provide corresponding equations for the cases in which a tape is *nil* yet expected to provide a cell; in this case, we assume the same behavior as if a zero cell were available:

$$q(L, nil) = q'(L, c : nil) \text{ if } b \text{ is 0 and } c \text{ is 0 or 1,}$$
$$q(L, nil) = q'(0 : L, nil) \text{ if } b \text{ is 0 and } c \text{ is } \rightarrow,$$
$$q(b' : L, nil) = q'(L, b' : nil) \text{ if } b \text{ is 0 and } c \text{ is } \leftarrow,$$
$$q(nil, nil) = q'(nil, nil) \text{ if } b \text{ is 0 and } c \text{ is } \leftarrow,$$
$$q(nil, b : R) = q'(nil, 0 : b : R) \text{ if } c \text{ is } \leftarrow.$$

Let $\mathcal{E}_M$ be the equational specification above and let $\mathcal{R}_M$ be the corresponding TRS when all the equations are regarded as rewrite rules, oriented from left to right.

PROPOSITION 2. *The TRS $\mathcal{R}_M$ is orthogonal, so confluent, and the following are equivalent:*

(1) The Turing machine M terminates on input $b_1 b_2 \ldots b_n$;
(2) The term $q_s(nil, b_1 : b_2 : \cdots : b_n : nil)$ reduces to 1 in $\mathcal{R}_M$;
(3) The ground equality $q_s(nil, b_1 : b_2 : \cdots : b_n : nil) = 1$ can be derived using $\mathcal{E}_M$.

Proof: Since for any $q \in Q$ different from q_h there is precisely one rule whose left-hand-side (lhs) has the form $q(L, 0 : R)$ and precisely one whose lhs has the form $q(L, 1 : R)$, it follows that the lhs-es of the rules in the first group (treating the situations in which the tapes are not *nil* when their first elements are requested) cannot overlap. Also, note that lhs-es of the rules in the first group cannot overlap with those in the second group, because the right list arguments are non-*nil* in the former while, except for the last, they are *nil* in the second; the last rule in the second group could only possibly overlap with the last rule in the first group, but that is not possible either because of their left list arguments (one is *nil* while the other is non-*nil*). Finally, note that the rules in the third group cannot overlap with each other either; the only ones which could possibly overlap are the last three, but the *nil* vs. non-*nil* characteristics of their list arguments exclude each other. Therefore, the $\mathcal{R}_M$ is orthogonal, so confluent.

It is clear that any computation in M can be seamlessly simulated by $\mathcal{R}_M$; indeed, the computation in M on an input $b_1 b_2 \dots b_n$ can be simulated *step-by-step* by $\mathcal{R}_M$, starting with the term $q_s(nil, b_1 : b_2 : \cdots : b_n : nil)$. Also, any rewrite sequence in $\mathcal{R}_M$ generates stepwise a corresponding computation in M, by simply concatenating the reversed left list with the right one, and replacing the two *nil*s by infinite streams of zeros. Consequently, M reaches its state q_h during a computation if and only if the corresponding rewriting sequence in $\mathcal{R}_M$ ends with a term of the form $q_h(L, R)$. The equivalence of *(1)* and *(2)* above follows from the fact that there is only one way to reduce the term $q_s(nil, b_1 : b_2 : \cdots : b_n : nil)$ to 1, namely reducing it to $q_h(L, R)$ and then in one step to 1, and the equivalence of *(2)* and *(3)* follows by the Church-Rosser property of equational logic and the confluence of $\mathcal{R}_M$. □

3.3 An Encoding Using Infinite Streams

The encoding of a Turing machine as an equational specification presented above was chosen in such a way to ensure that it becomes operational when equations are regarded as rewrite rules and applied unrestricted; in particular, the encoding of the infinite tapes as finite lists ensured that one can use any rewrite engine, with no restrictions or reduction strategies, to perform Turing machine computations. The price to pay for this was that empty lists had to be treated in a special way. We can give a more elegant encoding of a Turing machine if we assume some special equational infrastructure, that of infinite streams, together with a corresponding reduction strategy when equations are regarded as rewrite rules.

As in the previous encoding, let us consider an operation q : *Stream* $\times$ *Stream* $\rightarrow$ *Bit* for each state $q \in Q$ and let us "encode" each transition $\delta(q, b) = (q', c)$ in M where $q \neq q_h$ with one equation:

$$q(L, b : R) = q'(L, c : R) \text{ if } c \text{ is 0 or 1,}$$
$$q(L, b : R) = q'(b : L, R) \text{ if } c \text{ is } \rightarrow, \text{ or}$$
$$q(b' : L, b : R) = q'(L, b' : b : R) \text{ if } c \text{ is } \leftarrow.$$

To state that the computation ends when the state q_h is reached, we also add the equation

$$q_h(L, R) = 1.$$

Let $\mathcal{E}_M^\infty$ be the equational specification above and let $\mathcal{R}_M^\infty$ be the corresponding TRS when all the equations are regarded as rewrite rules, oriented from left to right, and reduction is "lazy".

PROPOSITION 3. *The TRS $\mathcal{R}_M^\infty$ is confluent and the following are equivalent:*

(1) The Turing machine M terminates on input $b_1 b_2 \dots b_n$;
(2) The term $q_s(zeros, b_1 : b_2 : \cdots : b_n : zeros)$ reduces to 1 in $\mathcal{R}_M^\infty$;
(3) The ground equality $q_s(zeros, b_1 : b_2 : \cdots : b_n : zeros) = 1$ can be derived using $\mathcal{E}_M^\infty$.

Proof: The proof is essentially the same as that of Proposition 2, except for the confluence of $\mathcal{R}_M^\infty$, which follows by noticing that all its critical pairs, formed by unifying the variables L or R with *zeros* or *ones*, are join-able. □

4. The Π_2^0-Completeness of Stream Equality

We only need to show the Π_2^0-hardness of the problem. We show it by reduction to a problem known to be Π_2^0-complete, described below. We used a similar technique in [3] to show the Π_2^0-hardness of behavioral equivalence in general, via a complex encoding not based on streams. From the perspective of behavioral equivalence, what makes the result below interesting is that it holds for streams, this very basic, canonical example of observational, behavioral, or coalgebraic definition.

4.1 The Totality Problem

We claim that there are some Turing machines M, such that the following problem, called TOTALITY:

INPUT: An integer $k \geq 0$;
OUTPUT: Does M halt on all inputs $1^j 0 1^k$ for all $j \geq 0$?

is Π_2^0-complete. It is obvious that TOTALITY is in Π_2^0 for any Turing machine M. To show that it is Π_2^0-hard, we may choose M to be a universal Turing machine such that on input $1^j 0 1^k$, M computes $f_k(j)$, where f_k is the (partial) function computed by Turing machine with Gödel number k under some canonical assignment of Gödel numbers to Turing machines. By appropriately choosing conventions for Turing machines, $f_k(j)$ is defined if and only if the Turing machine numbered k halts on input j. Therefore, TOTALITY(k) has positive solution if and only if the function f_k is total. But the set $\{k \mid f_k \text{ is total}\}$ is Π_2^0-complete [7]. It follows that TOTALITY is Π_2^0-complete.

We henceforth fix some choice of M that makes the TOTALITY problem Π_2^0-complete.

4.2 Equality of Streams is Π_2^0-Complete

We can now show that the problem of saying whether two equationally (finitely) presented infinite streams are equal is Π_2^0-complete. The membership to Π_2^0 part of the result was shown in Section 2 and followed by the completeness of the first-order logic of equality; the hardness part is shown by reduction to TOTALITY. To reduce the problem of equality of streams to the TOTALITY problem, we need to define for any integer $k \geq 0$ a pair of streams such that M halts on all inputs $1^j 0 1^k$ for all $j \geq 0$ if and only if the two streams are equal. We discuss two possible reductions in what follows. The first captures the essence of the difficulty of this problem using a minimal "non-standard" infrastructure, namely by defining only one stream operation: one generating a stream from a finite input. While from an algebraic perspective this reduction shows clearly the subtle role played by streams in the Π_2^0 hardness, from a coalgebraic perspective it has the drawback that it is based on a relatively complex algebraic infra-structure of list of bits. To eliminate any doubt that the hardness of the stream equality problem comes from its algebraic infrastructure, we give a second reduction based exclusively on streams of bits; this second reduction is purely coalgebraic, in the sense that the resulting encoding is a correct equational definition of streams in the particular model (coalgebra) of streams.

The first reduction builds upon the encoding in Section 3.2. In the context of $\mathcal{E}_M$, we can pick for any integer $k \geq 0$ the pair of streams

$$\mathit{total?}(0 : 1 : \cdots : 1 : \mathit{nil}) \stackrel{?}{=} \mathit{ones}$$

where there are k ones following the zero in the argument of *total?* and where *total?* is the operation taking a (finite) list to an (infinite) stream defined as follows:

$$\mathit{total?}(R) = q_s(\mathit{nil}, R) : \mathit{total?}(1 : R).$$

As expected, the second reduction builds upon the encoding in Section 3.3. In the context of $\mathcal{E}_M^\infty$, we can pick for any $k \geq 0$ the pair of streams

$$\mathit{total?}(0 : 1 : \cdots : 1 : \mathit{zeros}) \stackrel{?}{=} \mathit{ones}$$

where there are k ones following the zero in the argument stream of *total?* and where *total?* is the operation taking a stream to a stream defined as follows (now R is a stream variable):

$$\mathit{total?}(R) = q_s(\mathit{zeros}, R) : \mathit{total?}(1 : R).$$

We only show the correctness of the second reduction; the first one can be shown similarly.

THEOREM 2. *(Π^0_2-hardness) For a given $k \geq 0$, M halts on all inputs 1^j01^k for all $j \geq 0$ if and only if the equality of streams total?$(0 : 1 : \cdots : 1 : zeros)$=ones holds in $\mathcal{E}^\infty_M$, where there are k bits of 1 following the bit 0 in the argument stream of total?.*

Proof: Recall from Theorem 1 that for any streams *str* and *str'*, the equality *str* $=$ *str'* holds whenever $\gamma(str) = \gamma(str')$ can be proved ordinarily (i.e., using the complete derivation systems of FOL$_=$) from $\mathcal{E}^\infty_M$ for any "experiment" γ of the form $head(tail(\cdots tail(\star)\cdots))$, where the dots stay for an arbitrary number of *tail* operations and the star for the hole where the stream to experiment upon is placed. By the definition of *total?*, the equality of streams *total?*$(0 : 1 : \cdots : 1 : zeros)$=*ones* therefore holds if and only if $q_s(zeros, 1 : \cdots : 1 : 0 : 1 : \cdots : 1 : zeros) = 1$ for any arbitrary number of 1 bits before the 0 at the beginning of the second argument stream of q_s, which, by Proposition 3, is equivalent to saying that M halts on all inputs 1^j01^k, for all $j \geq 0$. □

COROLLARY 1. *Proving equality on streams defined equationally is a Π^0_2-complete problem.*

Proof: It follows by Theorems 1 and 2. □

5. Conclusion

We gave a precise characterization for the complexity of stream equality, namely Π^0_2. The membership to the class Π^0_2 followed by the completeness of first-order logic of equality, and the Π^0_2 hardness followed by an encoding of the totality problem for partially recursive functions as a stream equality problem. Since the Π^0_2 class includes properly both the recursively enumerable and the co-recursively enumerable classes, this result implies that one can find no mechanical procedure to say when two streams are equal, as well as no procedure to say when two streams are not equal. In particular, there is no complete proof system for equality of streams and no complete system for dis-equality of streams. Since streams form a canonical example of coinductive type, of coalgebra, of observational specification and of hidden logic theory, the result in this paper tells us that any complete deduction system for these frameworks would impose restrictions on the input theory and/or the task to be proved that may be unacceptable for many of us.

Acknowledgments

The motivation for this work came from several interesting discussions at the Infinity Symposium 2006 in Amsterdam on the relationship between infinite rewriting, coalgebra, coinduction, as well as their applications. In particular, it was not clear to us to what extent the Π^0_2-completeness result in [3] applied to coalgebra as well, or whether it was an artifact of the mixed algebraic and coalgebraic particularities of hidden logics. Since streams form a canonical example for all the approaches to behavioral equivalence, the idea of proving a Π^0_2-completeness result for streams took shape. The author would like to thank the organizers of and to the participants to the Infinity Symposium, in particular to (alphabetically) Henk Barendregt, Jörg Endrullis, Clemens Grabmayer, Jan Willem Klop, Lawrence Moss and Jan Rutten for discussions and hints that motivated this work. Also, special thanks to Traian Florin Şerbănuţă for pointing several simplifications in a previous draft of this paper.

References

[1] J. Bergstra and J. V. Tucker. Equational specifications, complete term rewriting systems, and computable and semicomputable algebras. *Journal of the Association for Computing Machinery*, 42(6):1194–1230, 1995.

[2] M. Bidoit, R. Hennicker, and A. Kurz. Observational logic, constructor-based logic, and their duality. *Theoretical Computer Science*, 3(298):471–510, 2003.

[3] S. Buss and G. Roşu. Incompleteness of behavioral logics. In H. Reichel, editor, *Proceedings of Coalgebraic Methods in Computer Science (CMCS'00), Berlin, Germany, March 2000*, volume 33 of *Electronic Notes in Theoretical Computer Science*, pages 61–79. Elsevier Science, 2000.

[4] H. Geuvers. Inductive and coinductive types with iteration and recursion. In B. Nordström, K. Pettersson, and G. Plotkin, editors, *Informal Proceedings Workshop on Types for Proofs and Programs, Båstad, Sweden, 8–12 June 1992*, pages 193–217. Dept. of Computing Science, Chalmers Univ. of Technology and Göteborg Univ., 1992.

[5] R. Hennicker. Context induction: a proof principle for behavioral abstractions. *Formal Aspects of Computing*, 3(4):326–345, 1991.

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

[7] H. Rogers Jr. *Theory of Recursive Functions and Effective Computability*. MIT press, Cambridge, MA, 1987.

[8] J. Rutten. A tutorial on coinductive stream calculus and signal flow graphs. *Journal of Theoretical Computer Science*, pages 443–481, Oct. 2005.