

Equality of Streams is a Π_2^0 -Complete Problem *

Grigore Roşu

Department of Computer Science,
University of Illinois at Urbana-Champaign
grosu@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_2^0 . Since the Π_2^0 class includes properly both the recursively enumerable and the co-recursively enumerable classes, this result implies that neither the set of pairs of equal streams nor the set of pairs of unequal streams is recursively enumerable. Consequently, one can find no algorithm for determining equality of streams, as well as no algorithm for determining inequality of streams. In particular, there is no complete proof system for equality of streams and no complete system for inequality of streams.

Categories and Subject Descriptors F.1.3 [Computation by Abstract Devices]: Complexity Measures and Classes—Reducibility and completeness; F.4 [Mathematical Logic and Formal Languages]; F.3.2 [Logics and Meanings of Programs]: Semantics

General Terms Theory, Languages

Keywords Streams, Algebraic specification, Infinite structures

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:

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

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

* Partly supported by NSF grants CCF-0448501 and CNS-0509321.

Streams can be formally defined many different, but ultimately equivalent ways; e.g., as a coinductive type [5], as a final coalgebra [14], as an observational specification [3] or as a hidden logic theory [10]. 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 many interesting properties of 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 $\text{zip}(\text{zeros}, \text{zeros}) = \text{zeros}$, $\text{zip}(\text{ones}, \text{ones}) = \text{ones}$ and $\text{zip}(\text{zeros}, \text{ones}) = \text{blink}$ 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 $\text{zip}(\text{zeros}, \text{zeros})$ 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 [14], *context induction* [8] in an observational equational framework, or *circular coinduction* [10] 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 $\text{zip}(\text{zeros}, \text{ones}) = \text{blink}$ as follows:

1. check that the two streams have the same head, 0;
2. take the tail of the two streams and generate the new goal $\text{zip}(\text{ones}, \text{zeros}) = 1 : \text{blink}$; this 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 $\text{zip}(\text{zeros}, \text{ones}) = \text{blink}$, which is nothing but the original proof task;
5. conclude that $\text{zip}(\text{zeros}, \text{ones}) = \text{blink}$ 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. More-

[copyright notice will appear here]

over, 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_2^0 -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 inequality of streams. Recall that Π_2^0 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 and x and y vary over r.e. domains [13].

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_2^0 -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. above is probably hard to achieve satisfactorily.

This paper is structured as follows. Section 2 defines streams formally and shows that the equality problem of streams belongs to the class Π_2^0 . 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 a problem known to be Π_2^0 -complete to the equality problem of streams, thus proving the remaining Π_2^0 -hardness result. Finally, Section 5 concludes the paper.

2. Streams and Membership in Π_2^0

As already mentioned, streams and equational definitions of streams can be formalized in many equivalent ways: as coalgebras [14], as observational specifications [3], and as behavioral, or hidden specifications [10], with only two observers for experiments, namely taking the head and the tail of a stream. 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.

2.1 Behavioral Definitions of Streams

The membership of the stream equality problem in the Π_2^0 class will follow 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

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

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{E} , 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 *unconditional* and *finite* equational theories: equational theories over a finite number of symbols and with a finite number of unconditional equations.

Let $\mathcal{E} = (\Sigma, E)$ be an equational definition of streams, 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 than 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{E} 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:

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

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:

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

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 $m = 0 : even(m)$? 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{E} iff it has the same elements in the same order in any model of \mathcal{E} . \square

A *model of streams* is any set A_{Stream} together with two functions $A_{head} : A_{Stream} \rightarrow \{0, 1\}$ and $A_{tail} : A_{Stream} \rightarrow A_{Stream}$. Given any model of streams $(A_{Stream}, A_{head}, A_{tail})$, we can define a behavioral equivalence as *indistinguishability under experiments with head and tail* as follows: $a, a' \in A_{Stream}$ are *behaviorally equivalent*, written $a \equiv a'$, iff $A_{\gamma}(a) = A_{\gamma}(a')$ for any $\{head, tail\}$ -experiment γ , i.e., a *head* followed by a finite number of *tail* operations, where $A_{head(tail(\dots tail(*)))}(a)$ is a shorthand for $A_{head}(A_{tail}(\dots A_{tail}(a)))$. 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 $A_{head}(a) = A_{head}(a')$ and $A_{tail}(a) \equiv A_{tail}(a')$),

and that it is the *largest* such congruence. One can easily show that models of streams are particular *hidden algebras* in the sense of [10], or particular *coalgebras* over a functor $Set \rightarrow Set$ that takes a set X to $\{0, 1\} \times X$ [14]. The reader need not be familiar with observational logic, or hidden algebra or coalgebra.

If Σ is a *signature over streams*, that is, a signature including sorts *Bit* and *Stream* and operations $_{-} : Bit \times Stream \rightarrow Stream$, $head : Stream \rightarrow Bit$ and $tail : Stream \rightarrow Stream$, then a Σ -*model of streams*, or simply a Σ -*model*, is a Σ -algebra A that protects the bits: A is a pair consisting of

- a family of sets (called *carriers*) $\{A_s \mid s \in Sorts(\Sigma)\}$, in particular a set of “streams” A_{Stream} , and of
- a family of functions $\{A_\sigma : A_{s_1} \times \cdots \times A_{s_n} \rightarrow A_s \mid \sigma : s_1 \times \cdots \times s_n \rightarrow s \in Operations(\Sigma)\}$,

with the following restrictions:

- $A_{Bit} = \{0, 1\}$,
- $A_0 = 0$, and
- $A_1 = 1$.

Note that any Σ -model of streams A includes a model of streams $(A_{Stream}, A_{head}, A_{tail})$. One would also like the Σ -models of streams to satisfy the equations at the beginning of this section:

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

However, the second one need only be satisfied behaviorally, so we postpone this requirement until we define behavioral satisfaction (right next). We ambiguously let \equiv denote the behavioral equivalence on A_{Stream} extended with identity relations on all the other sorts, including A_{Bit} . Let $str = str'$ be a Σ -equation over variables V and A be a Σ -model. Then A (*behaviorally*) *satisfies* the equation $(\forall V) str = str'$, written $A \models (\forall V) str = str'$, iff $\theta^*(str) \equiv \theta^*(str')$ for any (appropriate many-sorted) mapping $\theta : V \rightarrow A$, 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 $\mathcal{E} = (\Sigma, E)$ is an *equational specification of streams*, that is, if Σ is a signature over streams and E is a set of Σ -equations including the two above, 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 .

Example. Consider an equational specification of streams \mathcal{E} consisting of the equations in Section 1

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

together with a desired property

$$zip(zeros, ones) = blink,$$

say e . As discussed in Section 1, there is no way to prove that $\mathcal{E} \vdash e$ using standard equational reasoning, with or without induction. However, we can easily show that e is a behavioral consequence of \mathcal{E} , that is, $\mathcal{E} \models e$. Indeed, let A be a model of \mathcal{E} , that is, $A \models \mathcal{E}$, and let us show that $A \models e$, that is, that for any $\{head, tail\}$ -experiment γ , i.e., a *head* followed by a finite number of *tail* operations, $A_\gamma(zip(zeros, ones)) = A_\gamma(blink)$. It is easier to show by induction on the size of γ a more general property, namely that for any $\{head, tail\}$ -experiment γ ,

$$\begin{aligned} A_\gamma(zip(zeros, ones)) &= A_\gamma(blink) \text{ and} \\ A_\gamma(zip(ones, zeros)) &= A_\gamma(1 : blink). \end{aligned}$$

If the size of γ is 1, that is, if γ is $head(*)$, then both terms in the first equality above evaluate to 0 and both terms in the second equality evaluate to 1; all four equations in \mathcal{E} are needed to show these. If γ has the form $\gamma'(tail(*))$ for some smaller γ' , then one can first show that

$$\begin{aligned} A_\gamma(zip(zeros, ones)) &= A_{\gamma'}(zip(ones, zeros)), \\ A_\gamma(blink) &= A_{\gamma'}(1 : blink), \\ A_\gamma(zip(ones, zeros)) &= A_{\gamma'}(zip(zeros, ones)), \text{ and} \\ A_\gamma(1 : blink) &= A_{\gamma'}(blink), \end{aligned}$$

and then one can use the induction hypothesis to conclude that the desired properties hold. \square

We next define the *stream equality problem* formally, as a decision problem:

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

We will show that this problem belongs to the Π_2^0 class/degree in the arithmetic hierarchy, but we first need to prove some properties relating behavioral satisfaction to first-order logic satisfaction:

PROPOSITION 1. *Let $\mathcal{E} = (\Sigma, E)$ be a finite unconditional equational specification of streams, and let \mathcal{E}_{Bit} be the potentially infinite $FOL_{=}$ specification (i.e., it can have infinitely many finite formulae) defined as follows:*

- add to \mathcal{E}_{Bit} all the equations in \mathcal{E} of sorts different from *Stream*;
- add for each equation $(\forall V) str = str'$ of sort *Stream* in \mathcal{E} a recursively enumerable set of equations of sort *Bit*:
 $\{(\forall V) \gamma(str) = \gamma(str') \mid \gamma \text{ is a } \{head, tail\}\text{-experiment}\}$;
- add the formulae $\neg(0 = 1)$ and $(\forall B : Bit) B = 0 \vee B = 1$; these are the only non-equational formulae in \mathcal{E}_{Bit} .

Then the following hold:

- (1) For any Σ -algebra A , $A \models \mathcal{E}_{Bit}$ iff $A^{\{0,1\}}$ is a Σ -model of streams and $A^{\{0,1\}} \models \mathcal{E}$, where $A^{\{0,1\}}$ is A whose elements A_0 and A_1 in A_{Bit} are renamed by 0 and 1, respectively, and \models is the standard satisfaction in $FOL_{=}$;
- (2) If e is a Σ -equation of sort *Bit*, then $\mathcal{E} \models e$ iff $\mathcal{E}_{Bit} \models e$, where the latter is standard entailment in $FOL_{=}$.

Proof: (1) First note that for any Σ -algebra A , A satisfies the two non-equational formulae $\neg(0 = 1)$ and $(\forall B : Bit) B = 0 \vee B = 1$ in \mathcal{E}_{Bit} if and only if the carrier A_{Bit} of A has only two elements and those correspond to the constant operations 0 and 1, if and only if $A^{\{0,1\}}$ is a Σ -model of streams. Also, note that in this case, if e is some equation of sort different from *Stream* in \mathcal{E} , then $A \models e$ if and only if $A^{\{0,1\}} \models e$; that is because the behavioral equivalence relation is identity on all sorts different from *Stream*. All that is left to prove now is, also in the case above, that for any equation $(\forall V) str = str'$ of sort *Stream* in \mathcal{E} , the following are equivalent:

$$\begin{aligned} A \models \{(\forall V) \gamma(str) = \gamma(str') \mid \gamma \text{ is a } \{head, tail\}\text{-experiment}\}, \\ A^{\{0,1\}} \models (\forall V) str = str'. \end{aligned}$$

This equivalence follows from the observation that, for any mapping $\theta : V \rightarrow A$ and for any $\{head, tail\}$ -experiment γ , $\theta^*(\gamma(str))$ is nothing but $A_\gamma^{\{0,1\}}(\theta^{\{0,1\}*}(str))$, where θ^* is the extension of θ to Σ -terms and $\theta^{\{0,1\}} : V \rightarrow A^{\{0,1\}}$ is defined like θ in all variables in V of sort different from *Bit* and, for variables B of sort *Bit* in V , if any, $\theta^{\{0,1\}}(B) = 0$ iff $\theta(B) = A_0$ and $\theta^{\{0,1\}}(B) = 1$ iff $\theta(B) = A_1$; note that the mapping $\theta \rightsquigarrow \theta^{\{0,1\}}$ is a bijection $[V \rightarrow A] \xrightarrow{\sim} [V \rightarrow A^{\{0,1\}}]$.

(2) Let e be an equation of sort *Bit*, and let us first assume that $\mathcal{E} \models e$. To show that $\mathcal{E}_{Bit} \models e$, let us pick some Σ -algebra A such that $A \models \mathcal{E}_{Bit}$. By (1), it follows that $A^{\{0,1\}}$ is a Σ -model of streams and $A^{\{0,1\}} \models \mathcal{E}$, which implies that $A^{\{0,1\}} \models e$. Since e has a sort different from *Stream*, as explained in the proof of (1), it follows that $A \models e$. Therefore, $\mathcal{E}_{Bit} \models e$. Conversely, let $\mathcal{E}_{Bit} \models e$ and let A be any Σ -model of streams such that $A \models \mathcal{E}$. By the definition of Σ -models, A is a Σ -algebra; moreover, $A^{\{0,1\}}$ is nothing but A . By (1) it follows that $A \models \mathcal{E}_{Bit}$, which implies $A \models e$. Since e has sort *Bit*, it follows that $A \models e$. Hence $\mathcal{E} \models e$. \square

2.2 Membership in Π_2^0

Recall that Π_2^0 is the class, or the degree, 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 and x and y vary over r.e. domains, for simplicity natural numbers [13]. A typical Π_2^0 -complete problem is TOTALITY: giving a Turing machine M , does it terminate on all inputs? The complexity of the problem stays in the fact that there are infinitely many (but enumerable) inputs, so one can never be “done” with testing them; moreover, even for a given input, one does not know when to stop running the machine and reject the input. However, if one is given for each input accepted by the machine a run of the machine, then one can simply check the run and declare the input indeed accepted. Informally, in the case of TOTALITY, a ranges over Turing machines, P is the property saying that a Turing machine a terminates on all inputs, x ranges over all inputs, y over all runs (sequences of transitions and states of a), and R checks whether y is a correct run of the machine a on input x , a decidable procedure. This intuition will be formalized in Section 4.

To show that our stream equality problem is in Π_2^0 , we use a different and less standard argument, borrowed from [4]: given equational stream specification \mathcal{E} and equation $(\forall V)str = str'$, we show that for any $\{head, tail\}$ -experiment γ (a head followed by a finite number of tails, so experiments are r.e.), there is some proof π_γ (because of the complete deduction of $FOL_=$; proofs are also r.e.) that $\mathcal{E}_{Bit} \models (\forall V)\gamma(str) = \gamma(str')$ in $FOL_=$.

THEOREM 1. (Π_2^0 -membership) *For any equational stream definition $\mathcal{E} = (\Sigma, E)$ and any Σ -equation $(\forall V)str = str'$ of sort *Stream*, $\mathcal{E} \models (\forall V)str = str'$ if and only if $\mathcal{E}_{Bit} \models (\forall V)\gamma(str) = \gamma(str')$ in $FOL_=$ for all $\{head, tail\}$ -experiments γ . In particular, the stream equality problem is in Π_2^0 .*

Proof: The first part follows by (2) in Proposition 1, noticing that $\mathcal{E} \models (\forall V)str = str'$ if and only if $\mathcal{E} \models (\forall V)\gamma(str) = \gamma(str')$ for all $\{head, tail\}$ -experiments γ , the latter following immediately from the definition of \models . For the membership in the Π_2^0 class part we use the fact that \models in $FOL_=$ admits complete deduction: $\mathcal{E} \models (\forall V)str = str'$ if and only if for any experiment γ , there is some proof π_γ in $FOL_=$ of the entailment $\mathcal{E}_{Bit} \models (\forall V)\gamma(str) = \gamma(str')$. If one wants to be very rigorous, one can encode all experiments γ as natural numbers x and all proofs in $FOL_=$ as natural numbers y . Then one can more easily see that our stream equality problem is in Π_2^0 because it is a problem of the form $P(a) := (\forall x)(\exists y)R(a, x, y)$, where a ranges over pairs of finite unconditional equational specifications of streams and equations of streams (\mathcal{E}, e) , P is the predicate that the equation specification (\mathcal{E}) entails the equation (e) , x ranges over experiments, y over $FOL_=$ proofs, and R is a procedure checking that y is a correct proof that the equational specification (\mathcal{E}) entails the experiment x applied to

the equation (e) . Since checking a given first-order proof is a decidable problem, the stream equality problem is in class Π_2^0 . \square

Note that we have under-used the completeness of $FOL_=$ in the proof of membership in Π_2^0 above, because the $FOL_=$ specification \mathcal{E}_{Bit} has only two non-equational formulae, namely the negation and the disjunction in Proposition 1. When proving membership in a 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_2^0 -membership result above holds for a larger class of stream definitions than just equational, but we do not know how far one can go. However, as shown in [10], 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 infinitely many experiments to check whether a conditional equation apply; in particular, the results in Proposition 1 do not hold.

We will show in Section 4 that the equality problem of two equationally defined streams is actually Π_2^0 -complete. The hardness part of the proof is by reduction from a problem known to be Π_2^0 -complete, namely TOTALITY.

2.3 On Models of Streams

The problem of defining the *right* models for streams seems to be non-trivial. We do not intend to solve this problem here, but only discuss some of its subtleties and argue that the models that we are using are reasonable. The notion of model appears to be crucial for the existence of a result in the style of 2 in Proposition 1, that is, the existence of some r.e. specification \mathcal{E}_{Bit} in $FOL_=$ or any other logic including equality and admitting complete deduction, such that $\mathcal{E} \models (\forall V)str = str'$ if and only if $\mathcal{E}_{Bit} \models (\forall V)\gamma(str) = \gamma(str')$ for all $\{head, tail\}$ -experiments γ ; this result is essential for the Π_2^0 -membership result. In particular, for some notions of models of streams discussed next we do not know whether the stream equality problem is in Π_2^0 or not. The Π_2^0 -hardness result in the next sections is so basic that it works regardless of the particular notion of model of streams chosen; the reader who is not interested in the semantic aspects of streams may skip the remaining of this section.

Let us first discuss our current models informally. In our models, streams are defined as “blackboxes” being able to generate, when requested with *head* and *tail*, infinitely many elements. A model is a universe of such blackbox streams, in which two streams are declared equivalent if and only if they can produce the same elements when requested with $\{head, tail\}$ -experiments. Operations on streams different from *head* and *tail* can be seen as constructors of blackboxes, the same way one can build circuits by connecting other circuits; for example, $zip(S, S')$ returns *head*(S) when requested for its head, and transits to blackbox $zip(S', tail(S))$ when requested a tail. This way, $zip(S, S')$ “knows” how to produce any of its elements when requested, by appropriately inquiring S and S' for their heads or tails. In our view, these are the most natural models, but they are still open for debate for two reasons:

1. some streams may not have corresponding blackboxes in some models; in particular, a model that satisfies most equational specifications is a degenerated one which has only one element in its carrier (all equations of sort *Stream* will hold in such a model, though some equalities of sort *Bit* may not hold); and
2. the interpretations of operations in some models may not be *behaviorally congruent*, in the sense that they may not take behaviorally equivalent streams to behaviorally equivalent ones.

Therefore, one could say that we allow “too many” models of streams. Nevertheless, one typically wants to be semantically permissive when one proves upper bound results, such as our membership to Π_2^0 . Similarly, one may argue that equational logic al-

lows too many models, including especially the degenerated (one-element carrier) ones; unfortunately, without such models one cannot show the completeness result of equational deduction.

Regarding the second reason above, namely the behavioral incongruence of some operators in some models, we believe that the way one should disallow such models if one does not want them is to constrain the models semantically, by adding appropriate equations that define the meaning of all operations unambiguously. For example, one can prove that the *zip* operation as defined in this paper is behaviorally congruent in all models. Proof theoretical *congruence criteria* are discussed in [12]. One may reject all equational specifications of streams that do not pass such “well-definedness” criteria; the well-definedness problem for stream equational definitions is probably itself a Π_2^0 problem, but the details still need to be worked out. However, there are situations in which one may want to allow incongruent operations. Consider, for example, a variant of the “nondeterministic stack” in [7] adapted to streams: one wants to define a non-deterministic operation $push : Stream \rightarrow Stream$ that non-deterministically adds a 0 or a 1 to the head of a stream. One can define this operation by adding just one equation

$$(\forall S) tail(push(S)) = S$$

saying that *push* constructs a new bit to the head of a stream, but does not modify any of the existing bits in the stream. Such an operation on bits may be desirable if one wants to define or reason about a random bit generator. The actual streams of bits can be regarded as “behavioral projections” of system’s executions. In particular, they do not show the actual current state (e.g., the seed) of the generator; the state of the generator is obviously kept as part of the “blackbox”, but is not (desired to be) retrievable with $\{head, tail\}$ -experiments. This way, blackboxes that are behaviorally equivalent may still behave differently under a *push* operation. If *push* was required to be behaviorally congruent, then one would exclude all the interesting models of such a specification, because one wants a random bit generator’s next bit not to depend on the already generated bits. In particular, if *push* was behaviorally congruent then the following equality would hold behaviorally,

$$(\forall S) push(tail(push(S))) = push(S),$$

meaning that in any model, if one generates a new bit, then removes it (for example, a client process may consume it) and then generate another bit, the second bit will be the same as the first one. This is clearly an undesirable behavior of a random bit generator.

An approach alternative to allowing behaviorally incongruent operations could be to define behavioral equivalence using *all* the operations in the specification, not only *head* and *tail*, like in the original setting of hidden algebra [6]. A Π_2^0 -completeness result for equational behavioral entailment has been shown in [4] for the general setting of hidden algebra, from which we get immediately the Π_2^0 -membership of the stream equality problem with such stream models, because stream definitions with such models are special instances of hidden algebra theories [10]. In fact, the technique used in this paper to show the membership to Π_2^0 (for any experiment, there is a proof ...) is taken over from [4]. However, we believe that requiring all the operations participate in the definition of the behavioral equivalence of streams is acceptable only when they generate the same behavioral equivalence as head and tail, which is equivalent to saying that all operations are behaviorally congruent for the behavioral equivalence generated by head and tail [12]. Therefore, under the well-definedness restriction of stream equational definitions, the models of streams obtained following this approach are precisely the same as the models considered in this paper. However, if well-definedness is not required, such as the case of the nondeterministic *push* above, then there is no inclusion relation between our current models of streams and the models of

this alternative approach. In particular, our Π_2^0 -membership result in this paper does *not* follow from the Π_2^0 result in [4].

Another semantic approach to streams could be to drop the view of “streams as blackboxes” entirely and instead to consider only models whose carriers contain precisely all the streams of bits, that is, functions from natural numbers to $\{0, 1\}$. In other words, a model would consist just of interpretations of operations as functions on actual streams that satisfy all the equational constraints, with no reference to its carrier, because that is understood: all the streams of bits. We find such a simplistic semantic approach limited, because it would not allow one to define subtypes of streams elegantly. For example, one would like to be able encode natural numbers with infinity as streams by first defining an operation $nat? : Stream \rightarrow \{0(false), 1(true)\}$ saying whether a stream has only ones (1^ω , encoding ∞) or a finite number of ones followed only by zeros ($1^n 0^\omega$, encoding n), and then adding the equation

$$(\forall S) nat?(S) = 1.$$

If one requires models to contain all streams in their carriers, then such an elegant definition of natural numbers using streams would be inconsistent. Indeed, the equation above does not hold; in particular, the stream 0101... violates it. Nevertheless, it would also be interesting to study the complexity of the stream equality problem for such models. The results that follow in the paper give Π_2^0 as a lower bound, but we do not know any upper bound.

A relaxation to the above strict notion of model is to allow models whose carriers are subsets of all streams. This way, the specification above would allow the desired model that is isomorphic to natural numbers with infinity. While it would be interesting to find an upper bound for the stream equality problem with such models, we believe that this approach is still limited, since it has “too few” models. In particular, one would disallow the desirable models for the nondeterministic *push* above. Yet another relaxation of these models is to allow operations to be nondeterministic, that is, to interpret them in models as relations over a subset of streams, rather than as functions. In this case, we believe that these models are semantically equivalent with our current models, so we conjecture the stream equality problem to be in Π_2^0 for such models, too.

This section suggests that a systematic study of models of streams, with their advantages, disadvantages and complexities, would be nevertheless interesting. However, this is not our goal in this paper. We believe that our current notion of model is natural and intuitive enough to claim the membership of the stream equality problem to the Π_2^0 class/degree. The rest of the paper is dedicated to showing the Π_2^0 -hardness of the stream equality problem, which holds for all semantic approaches discussed in this section.

3. Encoding Computation by Equational Deduction and Rewriting

Equational encodings of general computation into equational deduction are well-known; for example, [2, 1] show such encodings, 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_2^0 -hardness of the stream equality problem. While we could have used existing encodings of Turing machines as TRSs, however, we found them more complex and intricate for our purpose in this paper than needed. Consequently (and also for the sake of self-containment), we provide novel (simple) encodings and corresponding proofs in this paper. Since the subsequent encodings are general purpose rather than specific to our Π_2^0 -hardness result, the content of this section may have a more pedagogical than technical nature. In particular, Sec-

tion 3.2 is not needed for the main result in Section 4 (Theorem 2), but it helps understand and motivate the encoding in Section 3.3. Also, the references to TRSs are technically only needed to prove the equational encoding correct, so they could have been removed from the main text and added only in the proofs, but we find them pedagogically interesting and potentially useful for other purposes.

The classic encodings in [2, 1] aim at emphasizing the strength of equational reasoning or the general undecidability of termination of term rewriting; they are not intended to be elegant or compact. Since we believe that the Π_2^0 -hardness result can be useful in other settings as well, for example in the context of infinitary rewriting [9], 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; in other words, there are no artificial rewrite steps. 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 [13], 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, 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 . For convenience, we write the left tape L backwards, that is, its head is at its right end; for example, Lb cons a b to the left tape L .

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 exam-

ined cell contains b , and if $M(q, b) = (q', c)$, 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 $(q_s, \dots 0 \dots 0|0 \dots 0 \dots)$. Sometimes, we wish to run a Turing machine on a specific input, say $x = b_1 b_2 \dots b_n$. In this case, its initial configuration is $(q_s, \dots 0 \dots 0|b_1 b_2 \dots b_n 0 \dots 0 \dots)$. 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 [13]. 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 $M(q, b) = (q', c)$ in where $q \neq q_h$ with precisely one equation, as follows:

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

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 have to define the same behavior as if a zero cell were available; in other words, we “generate” fresh tape (writing zeros on it) on a by-need basis:

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

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 $b_1 b_2 \dots b_n$;*
- (2) *The term $q_s(nil, b_1 : b_2 : \dots : b_n : nil)$ reduces to 1 in \mathcal{R}_M ;*

(3) The ground equality $q_s(\text{nil}, b_1 : b_2 : \dots : b_n : \text{nil}) = 1$ 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 second 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, \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 $q_s(\text{nil}, b_1 : b_2 : \dots : b_n : \text{nil})$. 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 $q_s(\text{nil}, b_1 : b_2 : \dots : b_n : \text{nil})$ 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 . \square

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 unrestrictedly; 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 $q : \text{Stream} \times \text{Stream} \rightarrow \text{Bit}$ for each state $q \in Q$ and let us “encode” each transition $M(q, b) = (q', c)$ where $q \neq q_h$ with one equation:

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

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^∞ 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, and reduction is “lazy”.

PROPOSITION 3. *The TRS \mathcal{R}_M^∞ 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 $q_s(\text{zeros}, b_1 : b_2 : \dots : b_n : \text{zeros})$ reduces to 1 in \mathcal{R}_M^∞ ;

(3) The ground equality $q_s(\text{zeros}, b_1 : b_2 : \dots : b_n : \text{zeros}) = 1$ can be derived using \mathcal{E}_M^∞ .

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

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 reducing to it a problem known to be Π_2^0 -complete, described below. We used a similar technique in [4] 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 $1^j 01^k$ 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 $1^j 01^k$, 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 [13]. 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 now show that the problem of saying whether two equationally (finitely) presented infinite streams are equal is Π_2^0 -complete. The membership in 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 from TOTALITY. To reduce the TOTALITY problem to the problem of equality of streams, we need to define for any integer $k \geq 0$ a pair of streams such that M halts on all inputs $1^j 01^k$ 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 infrastructure 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 pick for any integer $k \geq 0$ the pair of streams

$$\text{total?}(0 : 1 : \dots : 1 : \text{nil}) \stackrel{?}{=} \text{ones}$$

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:

$$total?(R) = q_s(nil, R) : total?(1 : R).$$

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

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

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):

$$total?(R) = q_s(zeros, R) : total?(1 : R).$$

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

THEOREM 2. (Π_2^0 -*hardness*) *For a given $k \geq 0$, M halts on all inputs $1^j 0 1^k$ for all $j \geq 0$ if and only if the equality of streams $total?(0 : 1 : \dots : 1 : zeros) = ones$ holds in \mathcal{E}_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}_M^∞ for any “experiment” γ of the form $head(tail(\dots tail(\star)\dots))$, 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 $total?(0 : 1 : \dots : 1 : zeros) = ones$ therefore holds if and only if $q_s(zeros, 1 : \dots : 1 : 0 : 1 : \dots : 1 : zeros) = 1$ 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 $1^j 0 1^k$, for all $j \geq 0$. \square

COROLLARY 1. *Proving equality on streams defined equationally is a Π_2^0 -complete problem.*

Proof: It follows by Theorems 1 and 2. \square

5. Conclusion

We gave a precise characterization for the stream equality problem, namely Π_2^0 . The membership in the class Π_2^0 followed by the completeness of first-order logic of equality, and the Π_2^0 -hardness followed by an encoding of the totality problem for partially recursive functions as a stream equality problem. Since the Π_2^0 class includes properly both the recursively enumerable and the co-recursively enumerable classes, this result implies that neither the set of pairs of equal streams nor the set of pairs of unequal streams is recursively enumerable. Consequently, one can find no algorithm for determining equality of streams, as well as no algorithm for determining inequality of streams. In particular, there is no complete proof system for equality of streams and no complete system for inequality 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_2^0 -completeness result in [4] 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_2^0 -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, to Andrei Popescu for finding a mistake in the proof of Π_2^0 -membership in [11], which led to the discussion on models of streams in Section 2.3, and to the five anonymous referees.

References

- [1] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, New York, NY, USA, 1998.
- [2] 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.
- [3] M. Bidoit, R. Hennicker, and A. Kurz. Observational logic, constructor-based logic, and their duality. *Theoretical Computer Science*, 3(298):471–510, 2003.
- [4] 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.
- [5] 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.
- [6] J. Goguen. Types as theories. In G. M. Reed, A. W. Roscoe, and R. F. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 357–390. Oxford, 1991.
- [7] J. Goguen and G. Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, 2000.
- [8] R. Hennicker. Context induction: a proof principle for behavioral abstractions. *Formal Aspects of Computing*, 3(4):326–345, 1991.
- [9] R. Kennaway, J. W. Klop, M. R. Sleep, and F.-J. de Vries. Transfinite reductions in orthogonal term rewriting systems. *Inf. Comput.*, 119(1):18–38, 1995.
- [10] G. Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.
- [11] G. Roşu. Equality of streams is a Π_2^0 -complete problem. Technical Report UIUCDCS-R-2006-2708, University of Illinois at Urbana-Champaign, Department of Computer Science, Apr. 2006.
- [12] G. Roşu and J. Goguen. Hidden congruent deduction. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*. Springer, 2000. Lecture Notes in Artificial Intelligence, Volume 1761.
- [13] H. Rogers Jr. *Theory of Recursive Functions and Effective Computability*. MIT press, Cambridge, MA, 1987.
- [14] J. Rutten. A tutorial on coinductive stream calculus and signal flow graphs. *Journal of Theoretical Computer Science*, pages 443–481, Oct. 2005.