# On the Complexity of Equivalence of Specifications of Infinite Objects

Jörg Endrullis Dimitri Hendriks Rena Bakhshi

VU University Amsterdam
Department of Computer Science
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
{j.endrullis, r.d.a.hendriks, r.r.bakhshi}@vu.nl

## Abstract

We study the complexity of deciding the equality of infinite objects specified by systems of equations, and of infinite objects specified by $\lambda$-terms. For equational specifications there are several natural notions of equality: equality in all models, equality of the sets of solutions, and equality of normal forms for productive specifications. For $\lambda$-terms we investigate Böhm-tree equality and various notions of observational equality. We pinpoint the complexity of each of these notions in the arithmetical or analytical hierarchy.

We show that the complexity of deciding equality in all models subsumes the entire analytical hierarchy. This holds already for the most simple infinite objects, viz. streams over $\{0, 1\}$, and stands in sharp contrast to the low arithmetical $\Pi^0_2$-completeness of equality of equationally specified streams derived in [17] employing a different notion of equality.

***Categories and Subject Descriptors*** F.3.2 [*Logics and Meanings of Programs*]: Semantics; F.1.3 [*Computation by Abstract Devices*]: Complexity Measures and Classes; F.1.1 [*Models of Computation*]; F.4 [*Mathematical Logic and Formal Languages*]

***Keywords*** Infinite objects, equational specifications, lambda terms, equality, semantics, complexity.

## 1. Introduction

In the last two decades interest has grown towards infinite data, as witnessed by the application of type theory to infinite objects [5], as well as the emergence of coalgebraic techniques for infinite data types like streams [19], infinitary term rewriting and infinitary lambda calculus [24]. In functional programming, the use of infinite data structures dates back to 1976, see [11, 14].

We are concerned with the complexity of deciding the equality of infinite objects specified by systems of equations, and infinite objects specified by $\lambda$-terms. The equational specification of infinite objects is common practice in coalgebra, term rewriting and functional programming. Consider an example from [17]:

$$\left.\begin{array}{ll} \mathsf{zeros} = 0 : \mathsf{zeros} & \mathsf{ones} = 1 : \mathsf{ones} \\ \mathsf{blink} = 0 : 1 : \mathsf{blink} & \mathsf{zip}(x : \sigma, \tau) = x : \mathsf{zip}(\tau, \sigma) \end{array}\right\} \quad (1)$$

This is an equational specification of three infinite lists of bits, and a binary function over infinite lists.[1] Then, a typical question is whether the following equality holds:

$$\mathsf{zip}(\mathsf{zeros}, \mathsf{ones}) = \mathsf{blink} \quad (2)$$

The answer depends on the semantics we choose to interpret the equality; for example (2) is not valid in the hidden models considered in [17]; for more details we refer to Section 2. In order to answer such a question, we first need to settle on the precise semantics of equality for equational specifications; the candidates we consider in this paper are

I. Equality in all models.

II. Equality of the set of solutions.

For $\lambda$-terms we are not concerned with equality in the sense of convertibility (which is known to be $\Pi^0_2$-complete, see [1]). Instead, we are interested in *behavioral* equivalence of $\lambda$-terms in all contexts, because this corresponds to the interchangeability of expressions in purely functional languages. It is also closely related to referential transparency, and the notion of Böhm trees as values of expressions including those without normal form. Thus we consider the following equivalences for $\lambda$-terms:

III. Observational equivalences.

IV. Böhm-tree equality.

The 'right' choice of equivalence depends on the intended application. The classic semantics mentioned in items I and II above, are defined by model-theoretic means. From a algebraic perspective I and II are the most natural semantics to consider for equational reasoning. On the other hand, III and IV, are defined by means of evaluation, i.e., rewriting. In functional programming the latter are of foremost importance, because these take (lazy) evaluation strategies into account. From an evaluation perspective, two terms are equal if they have the same observable behavior, independent of the context they are in. In contrast to the model-theoretic notions, this equality is invariant under the exchange of *meaningless subterms*, that is, subterms which cannot be evaluated to a (weak) head

[0] The technical report supporting this paper can be found on `arxiv.org` under the same title.



[1] In Haskell there is $\mathsf{zip} :: [a] \to [b] \to [(a, b)]$, but we prefer to use 'zip' for the *interleaving* of lists, as defined by the equation in (1), since that is what a zipper does: it interleaves rows of teeth.

normal form. Another candidate for the semantics of equality is

V. Equality of normal forms for productive specifications.

A rewrite specification is *productive* [9, 23] if the terms under consideration can be fully evaluated, that is, (outermost-fair) rewriting yields a (possibly infinite) constructor normal form in the limit. In such a setting, equality of the normal forms is a suitable semantics for the equivalence of terms. Deciding the equality of productive specifications has been shown to be a $\Pi^0_1$-complete problem in [13]; this semantics is not considered here.

We now briefly describe the concepts I–IV.

***Equality in models (I and II).*** The semantics I (equality in all models) is useful when the objects under consideration are specified in the same specification. This semantics interprets the objects simultaneously in each model satisfying the specification. This allows us to compare objects that depend on a common unknown, an underspecified object; see (4) below for an illustrating example. If the objects under consideration are fully specified, that is, have unique solutions, then semantics I coincides with semantics II.

In contrast to I, semantics II is more suitable for comparing objects specified by different specifications, as we explain below. The objects are compared via the set of their solutions (in their respective specifications). This semantics is well-known from equations over real (or complex) numbers, where two equations, like

$$(x-1)^2 - 1 = 0 \qquad \text{and} \qquad x^2 - 2x = 0\,,$$

are equivalent if they have the same solutions for $x$, here $\{0, 2\}$.

A $\Sigma$-algebra $\mathcal{A}$ consists of a carrier set $A$ (the domain of $\mathcal{A}$) and an interpretation $[\![\cdot]\!]$ of the symbols $\Sigma$ occurring in the equational specification as functions over $A$. Then $\mathcal{A}$ is called a *model* of an equational specification $E$, which we denote by $\mathcal{A} \models E$, if all equations of $E$ respect the interpretation; that is, for every equation of $E$ both sides have the same interpretation for every assignment of the variables. As the domain we will typically choose (a subset of) the *final coalgebra* [21] describing the class of objects we are specifying. The final coalgebra ensures that the model is *continuous*, that is, if we have a converging sequence of terms $t_1, t_2, \ldots$ with limit $t_\omega$, then the sequence of interpretations $[\![t_1]\!], [\![t_2]\!], \ldots$ converges towards $[\![t_\omega]\!]$. For example, in a specification like

$$\mathsf{ones} = 1 : \mathsf{ones} \qquad\qquad \mathsf{ones}' = 1 : \mathsf{ones}' \tag{3}$$

the symbols $\mathsf{ones}$ and $\mathsf{ones}'$ are guaranteed to have the same interpretation. Continuity is crucial to conclude the validity of equations such as $\mathsf{ones} = \mathsf{ones}'$ which are not satisfied in non-continuous models like the *initial* algebra of the specification.

Let $E$ be a specification of $M$ and $N$. Then $M$ is considered equal to $N$ with respect to semantics I if every model of $E$ is also a model of $M = N$: $\forall \mathcal{A}.\ \ \mathcal{A} \models E \ \Rightarrow\ \mathcal{A} \models M = N$ . This notion is especially of interest if $M$ and $N$ depend on a common unknown and consequently have to be interpreted simultaneously in the same model. For example in

$$\left.\begin{array}{ll} M = \mathsf{zip}(X, X) & \mathsf{zip}(x:\sigma, \tau) = x : \mathsf{zip}(\tau, \sigma) \\ N = \mathsf{dup}(X) & \mathsf{dup}(x:\sigma) = x : x : \mathsf{dup}(\sigma) \end{array}\right\} \tag{4}$$

the streams $M$ and $N$ are both specified in terms of an unspecified stream $X$. Whatever interpretation $X$ has, $M$ and $N$ are equal, and so they are equal in the sense of semantics I.

On the other hand, semantics I has the effect that an underspecified constant is not equivalent to its renamed copy. This is illustrated by the following specification:

$$M = 0 : \mathsf{tail}(M) \qquad\qquad N = 0 : \mathsf{tail}(N) \tag{5}$$

Here $M$ and $N$ are not equal in every model; for example, let $[\![M]\!] = 0 : 0 : \ldots$ and $[\![N]\!] = 0 : 1 : 1 : \ldots$. Nevertheless, $M$ and $N$ are equal in the sense that they exhibit the same behaviors. That is, they have the same set of solutions: every stream starting with a zero is a solution for $M$ as well as for $N$. Thus, $M$ and $N$ are equal with respect to the semantics II. This paves the way for comparing objects $M$ and $N$ that are given by separate specifications $E_M$ and $E_N$, respectively. Note that it is not always suitable to apply semantics I to the union $E_M \cup E_N$ even if the specifications have disjoint signatures (using renaming), see further Remark 2.

Two objects $M$ and $N$ are equal with respect to semantics II if the set of solutions of $M$ in $E_M$ coincides with the set of solutions of $N$ in $E_N$: $\{\, [\![M]\!]^{\mathcal{A}} \mid \mathcal{A} \models E_M \,\} = \{\, [\![N]\!]^{\mathcal{A}} \mid \mathcal{A} \models E_N \,\}$. Here the set of solutions of a constant $X$ in a specification $E_X$ is the set of interpretations of $X$ in all models of $E_X$.

***Observational equivalence (III and IV).*** In purely functional languages based on the $\lambda$-calculus [1], the evaluation of expressions is *free of side effects*. As a consequence, an expression (or subexpression) can always be replaced by its normal form, the so-called *value* of the expression. This principle is known as *referential transparency*. This also implies that expressions can be substituted for each other if they have the same normal form.

For specifications of coinductive objects, such as infinite lists (called *streams*) or infinite trees, the value typically is an infinite term. For example in $\mathsf{ones} = 1 : \mathsf{ones}$, the term $\mathsf{ones}$ has as value (or infinite normal form) the infinite term $1 : 1 : 1 : \ldots$. However, it is not always guaranteed that a term can be fully evaluated. During the evaluation to the (possibly infinite) normal form, we may encounter subterms that cannot be evaluated because these subterms do not have a head normal form. In $\lambda$-calculus, such terms are known as *meaningless terms*. For example, consider:

$$\begin{array}{ll} \mathsf{natsx}(n) = n : \mathsf{g}(0) : \mathsf{natsx}(n+1) & \mathsf{g}(n) = \mathsf{g}(n) \\ \mathsf{natsx}'(n) = n : \mathsf{g}(n) : \mathsf{natsx}'(n+1) & \end{array}$$

Here $\mathsf{g}(n)$ is meaningless for every $n$. Consequently, $\mathsf{natsx}(0)$ evaluates to a stream in which every second element is meaningless, and therefore, undefined. An infinite value containing undefined parts can be represented by means of Böhm trees [1] introduced in 1975 by Corrado Böhm. In particular, the Böhm tree of $\mathsf{natsx}(0)$ is: $0 : \bot : 1 : \bot : 2 : \bot : 3 : \bot : 4 : \bot : \ldots$, where $\bot$ is a special symbol representing an undefined element.

In $\lambda$-calculus (or orthogonal higher-order rewriting), terms with equal Böhm trees can be exchanged (for each other) without changing the meaning of the whole expression. In the specification above, $\mathsf{natsx}(0)$ and $\mathsf{natsx}'(0)$ have the same Böhm tree, and hence are interchangeable. In contrast, from the model-theoretic perspective $\mathsf{natsx}(0)$ and $\mathsf{natsx}'(0)$ are different. In every model of $\mathsf{natsx}(0)$ all elements at odd indexes coincide, whereas $\mathsf{natsx}'(0)$ admits models that assign different interpretations to these elements. From a rewriting as well as functional programming perspective, these differences are irrelevant as they concern undefined subterms.

There are several notions of infinite values, depending on what terms are considered meaningless, including Böhm trees, Lévy-Longo trees, Berarducci trees, $\eta$-Böhm trees, $\eta^\infty$-Böhm trees; see further [6]. The terms $\lambda x.xx$ and $\lambda x.x(\lambda z.xz)$, for instance, have distinct Böhm trees, but we may want to consider the terms *behaviorally*, or *observationally equivalent* as they are $\eta$-convertible. There are several natural concepts of *observational equivalence* for $\lambda$-calculus, where terms are considered *equivalent* if they yield the same observations in every context. To that end, we consider three forms of *observations*: normal forms (nf), head normal forms (hnf), and weak head normal forms (whnf). A *head normal form* is a $\lambda$-term of the form $\lambda x_1 \ldots \lambda x_n . y N_1 \ldots N_m$ with $n, m \geq 0$. A *weak head normal form* is an hnf or an abstraction, i.e., a whnf is a term of the form $xM_1 \ldots M_m$ or $\lambda x.M$. Each of the observations gives rise to an equivalence $=_{nf}$, $=_{hnf}$ or $=_{whnf}$, defined by

$$M =_{nf} N \ \text{ iff } \ (\forall C.\, C[M] \text{ has a nf iff } C[N] \text{ has a nf})$$

$$M =_{hnf} N \text{ iff } (\forall C.\, C[M] \text{ has a hnf iff } C[N] \text{ has a hnf})$$
$$M =_{whnf} N \text{ iff } (\forall C.\, C[M] \text{ has a whnf iff } C[N] \text{ has a whnf})$$

In fact, the equivalence $=_{nf}$ corresponds to $\eta$-Böhm trees, and $=_{hnf}$ to $\eta^{\infty}$-Böhm trees. For more details we refer to [6], where it is argued that $=_{whnf}$ corresponds to evaluation strategies used by lazy functional languages. If two expressions behave the same in every context, then no functional program can distinguish them.

***Contribution.*** We characterize for each of the semantics I–IV the complexity of deciding the equality of terms. For I and II we will focus on equational specifications of bitstreams, and for III and IV on behavioral equivalences of $\lambda$-terms and Böhm tree equality.

Each of these equivalences is undecidable, therefore we characterize their complexity by means of the arithmetical and analytical hierarchies, see Figure 1. The arithmetical hierarchy classifies the complexity of a problem $P$ by the minimum number of quantifier alternations in first-order formulas that characterize $P$. The analytical hierarchy extends this classification to second-order arithmetic, then counting the alternations of set quantifiers.

(A) It turns out that the complexities of deciding the equality in all models as well as the equality of the set of solutions subsume the entire arithmetical and analytical hierarchy when the domain of the models is the set of all streams, so-called *full* models, see Theorems 3 and 5. The idea of the proof is as follows. We translate formulas of the analytical hierarchy into stream specifications by representing $\forall$ set quantifiers by equations with variables. This simulates a quantification over all streams as the models are full, and the equations have to hold for all assignments of the variables. The $\exists$ set quantifiers are eliminated in favor of Skolem functions (here stream functions). The interpretation of the functions is determined by the model, and the question whether there exists a model corresponds to an existential quantification over all Skolem functions.

(B) & (C) If we admit models whose domain does not contain all streams, then the complexity of deciding equality drops to the level $\Pi^1_1$ of the analytical hierarchy for semantics I, and to $\Pi^1_2$ for II, see Theorems 1, and 7. The reason is that equations with variables no longer have to hold for all streams, but only for the streams that exist in the model. By the Löwenheim-Skolem theorem we obtain that if there exists a model, then there exists a countable model: from an uncountable model we construct a countable one, by taking the finitely many streams "of interest" and closing them under all functions in the model. Thus, it suffices to quantify over countable models for which one single set quantifier is enough.

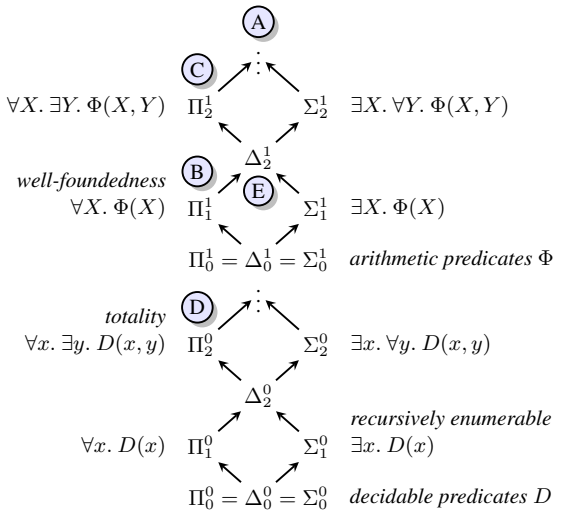


**Figure 1.** Arithmetical (bottom) and analytical hierarchy (top).

The aforementioned results already hold for bitstreams, one of the simplest coinductive objects, and thereby can serve as a lower bound on the hardness of the equality problem for other coinductive objects. We also study the behavioral semantics from [17]. We find that if behavioral equivalence $\equiv$ is required to be a congruence, like for example in [2], then the complexity of deciding behavioral equivalence is catapulted out of the arithmetical hierarchy, to the level $\Pi^1_1$ of the analytical hierarchy, see Theorem 8. Likewise so for the behavioural equivalence for specifications of streams of natural numbers, relaxing the congruence requirement, see Theorem 9.

(D) For the equivalences on $\lambda$-terms, we show that deciding the Böhm tree and Lévy–Longo tree equality, as well as the observational equivalences $=_{nf}$, $=_{hnf}$ and $=_{whnf}$ are $\Pi^0_2$-complete problems, see Theorem 11. (It is clear that when an object is given by a rewrite system, like the $\lambda$-calculus, then the complexity resides in the arithmetical hierarchy, since it suffices to quantify over a number steps to evaluate parts of the object.)

(E) Finally, we consider the complexity of unique solutions. A term $s$ has a unique solution within a specification $E$ if there exists models of $E$, and in all models of $E$, $s$ has the same interpretation. The problem of deciding unique solvability in all full models again subsumes the analytical hierarchy, see Theorem 4. When also considering the non-full models, we find that the problem is $\Pi^1_1$- and $\Sigma^1_1$-hard, but is strictly contained in $\Delta^1_2$, see Theorem 2.

***Outline.*** We first discuss related work. We formally introduce bitstream specifications and stream models in Section 3, and Turing machines with oracles in Section 4. We recall the basic complexity-related notions in Section 5. We use these concepts in Section 6 to derive the complexity results for the model-theoretic notions. In Section 7 we consider a different notion of models, namely the behavioural semantics as in [17]. Finally, we investigate the observational equivalences of $\lambda$-terms in Section 8.

## 2. Related Work

The complexity of the equality of streams specified by systems of equations has been investigated in the ICFP paper [17, Corollary 1]; we cite: *Proving equality on streams defined equationally is a* $\Pi^0_2$*-complete problem*. This result is based on a behavioral notion of stream models [16]. We briefly summarize the main characteristics of these models:

(i) Every stream $\sigma \in \{0,1\}^{\omega}$ can have multiple representatives in the model (known as *confusion*).

(ii) For every equation $\ell = r$ it is required that the interpretations $[\![\ell]\!]$ and $[\![r]\!]$ are *behaviorally equivalent*, denoted by $\equiv$, that is, equality under all $[\![\mathsf{head}]\!]([\![\mathsf{tail}]\!]^n(\Box))$ experiments. In particular, it is not required that $[\![\ell]\!] = [\![r]\!]$.

(iii) Behavioral equivalence $\equiv$ is not required to be a congruence.

Behavioral models have a wide range of applications, for example for modeling computations with hidden states, or capturing certain forms of nondeterminism. For these applications it is often intended that the semantics is not preserved under equational reasoning. For example, consider the following specification from [17]

$$\mathsf{tail}(\mathsf{push}(\sigma)) = \sigma\ ,$$

specifying a function push that prefixes an element to the argument stream, while leaving unspecified which element. In the behavioral models we obtain a restricted form of nondeterminism [25], for example, the following is not behaviorally satisfied:

$$\mathsf{push}(\mathsf{tail}(\mathsf{push}(\sigma))) = \mathsf{push}(\sigma)\ , \tag{6}$$

although derivable by pure equational reasoning. For a nondeterministic operation, it is of course desirable that (6) does not hold.

However, for function definitions employing pattern matching, behavioral models sometimes yield unexpected results; consider:

$$\mathsf{ones} = 1 : \mathsf{ones} \qquad \mathsf{f}(x:\sigma) = \sigma \tag{7}$$

Now, there are models that satisfy the specification (7), but not (8):

$$\mathsf{f}(\mathsf{ones}) = \mathsf{ones} \tag{8}$$

In these models we have that $[\![\mathsf{ones}]\!] \neq [\![1 : \mathsf{ones}]\!]$ and, at the same time, that $[\![\mathsf{ones}]\!]$ cannot be constructed by the stream constructor $[\![:]\!]$, that is, $[\![\mathsf{ones}]\!] \neq [\![:]\!](x, s)$ for all $x \in \{0,1\}$ and $s \in A_S$. Consequently, the interpretation $[\![\mathsf{f}]\!]([\![\mathsf{ones}]\!])$ can be arbitrary.

Thus, behavioral reasoning is typically not sound for behavioral models, and therefore the corresponding specifications are usually referred to as *behavioral specifications*. In this paper we are interested in specifications where equational reasoning is sound.

**Remark 1.** We construct a behavioral model $\langle A, [\![\cdot]\!]\rangle$ in the sense of [17] where specification (1) is behaviorally satisfied but the goal equation $\mathsf{zip}(\mathsf{zeros}, \mathsf{ones}) = \mathsf{blink}$ is not. The model thereby forms a counterexample to [17, Example 2].

We define the domain by $A_B = \{0,1\}$ and

$$A_S = \{z_w \mid w \in \{0,1\}^*\} \cup \{o_w \mid w \in \{0,1\}^*\} \cup \{0,1\}^\omega$$

Here $z_\varepsilon$ and $o_\varepsilon$ are alternative representations of $0^\omega$ and $1^\omega$, respectively, and $z_w$ and $o_w$ have an additional finite prefix $w \in \{0,1\}^*$. We define the interpretations $[\![\cdot]\!]$ for every $a \in \{0,1\}$, $\sigma \in \{0,1\}^\omega$, $w, v \in \{0,1\}^*$ and $x, y \in A_S$. For $[\![\mathsf{head}]\!]$ and $[\![\mathsf{tail}]\!]$ we define:

$$\begin{aligned}
[\![\mathsf{head}]\!](z_\varepsilon) &= 0 & [\![\mathsf{head}]\!](o_\varepsilon) &= 1 & [\![\mathsf{head}]\!](a\sigma) &= a\\
[\![\mathsf{tail}]\!](z_\varepsilon) &= z_\varepsilon & [\![\mathsf{tail}]\!](o_\varepsilon) &= o_\varepsilon & [\![\mathsf{tail}]\!](a\sigma) &= \sigma\\
[\![\mathsf{head}]\!](z_{aw}) &= a & [\![\mathsf{head}]\!](o_{aw}) &= a & &\\
[\![\mathsf{tail}]\!](z_{aw}) &= z_w & [\![\mathsf{tail}]\!](o_{aw}) &= o_w & &
\end{aligned}$$

We define the interpretation $[\![:]\!]$ of the stream constructor by:

$$[\![:]\!](a, z_w) = z_{aw} \qquad [\![:]\!](a, o_w) = o_{aw} \qquad [\![:]\!](a, \sigma) = a\sigma$$

Note that the elements $z_\varepsilon$ and $o_\varepsilon$ cannot be constructed by $[\![:]\!]$.

We interpret $[\![\mathsf{zeros}]\!]$, $[\![\mathsf{ones}]\!]$ and $[\![\mathsf{blink}]\!]$ as follows:

$$[\![\mathsf{zeros}]\!] = z_\varepsilon \qquad [\![\mathsf{ones}]\!] = o_\varepsilon \qquad [\![\mathsf{blink}]\!] = (01)^\omega$$

We define an auxiliary function $\bowtie$ that (similar to zip) interleaves the elements of finite or infinite words; for $u_1, u_2 \in \{0,1\}^{\leq\omega} = \{0,1\}^* \cup \{0,1\}^\omega$, let $au_1 \bowtie u_2 = a(u_2 \bowtie u_1)$ and $\varepsilon \bowtie u_2 = u_2$. We now define the interpretation $[\![\mathsf{zip}]\!]$ of the symbol zip as follows:

$$[\![\mathsf{zip}]\!](z_w, o_v) = \begin{cases} (w \bowtie v)0^\omega & \text{for } |w| = |v| \\ (w\ 0^\omega) \bowtie (v\ 1^\omega) & \text{otherwise} \end{cases}$$

$$[\![\mathsf{zip}]\!](o_w, z_v) = \begin{cases} (w \bowtie v)0^\omega & \text{for } |w| = |v| + 1 \\ (w\ 1^\omega) \bowtie (v\ 0^\omega) & \text{otherwise} \end{cases}$$

and in all other cases, we define $[\![\mathsf{zip}]\!](x, y) = emb(x) \bowtie emb(y)$ where $emb(z_w) = w0^\omega$, $emb(o_w) = w1^\omega$, and $emb(\sigma) = \sigma$.

It is straightforward to check that specification 1 is behaviorally satisfied, whereas $[\![\mathsf{zip}(\mathsf{zeros}, \mathsf{ones})]\!] = 0^\omega \not\equiv (01)^\omega = [\![\mathsf{blink}]\!]$.

The counterexample in Remark 1 employs the fact that the behavioral models of [17] do not require that every stream can be constructed by the (interpretation of the) stream constructor $[\![:]\!]$. As a consequence, the equation $\mathsf{zip}(x:\sigma, \tau) = x : \mathsf{zip}(\tau, \sigma)$ does not fully define $[\![\mathsf{zip}]\!]$; it defines $[\![\mathsf{zip}]\!](\sigma, \tau)$ only for those arguments $\sigma$ that can be constructed by $[\![:]\!]$.

The example illustrates that the behavioral models of [17] do not go along with function definitions using pattern matching. To fully define $[\![\mathsf{zip}]\!]$, we can specify it using the stream destructors: $\mathsf{head}(\mathsf{zip}(\sigma, \tau)) = \mathsf{head}(\sigma)$, and $\mathsf{tail}(\mathsf{zip}(\sigma, \tau)) = \mathsf{zip}(\tau, \mathsf{tail}(\sigma))$. This change of the specification format resolves the problem.

Alternatively, keeping the specification format, we can adapt the notion of models. To reestablish soundness of equational reasoning one can (i) exclude confusion or (ii) require that $\equiv$ is a congruence. Note that the common models of streams are free of confusion: final coalgebras [20], one-sided infinite words $A^\omega$, and the function space $\mathbb{N} \to A$. In hidden algebras [15], confusion is often allowed but its negative effects are prevented by restricting to behavioral models [2], in which behavioral equivalence is a congruence: $s \equiv t \Rightarrow f(\ldots, s, \ldots) \equiv f(\ldots, t, \ldots)$.

Our results show that when $\equiv$ is required to be a congruence (or confusion is eliminated), then the complexity of the equality of bitstreams that are specified equationally jumps from the low level $\Pi^0_2$ of the arithmetical hierarchy to the level $\Pi^1_1$ of the analytical hierarchy, thereby exceeding the arithmetical hierarchy. Moreover, we show that even for behavioral specifications with confusion (as in [17]), equality of *streams of natural numbers* is $\Pi^1_1$-complete. Consequently, the results of [17] are valid only for bitstreams in combination with the behavioral equality discussed above. For general behavioral specifications (not the special case of stream specifications), the $\Pi^1_1$-completeness has been shown in [3].

Term rewriting systems are closely related to equational specifications. The complexity of deciding various standard properties of term rewriting systems, such as productivity, termination and confluence (Church–Rosser), has been investigated in [8, 10].

## 3. Bitstream Specifications

We will focus mainly on *streams*, one-sided infinite sequences of symbols, the prime example of coinductive structures. There are various ways of introducing streams: as functions $\mathbb{N} \to A$ mapping an index $n$ to the $n$-th element of the stream, as final coalgebras over the functor $X \mapsto A \times X$, using coinductive types [12], or observational models [2]. All these definitions are equivalent in the sense that the resulting coalgebras are isomorphic.

For the model-theoretic semantics of equality, we will focus on specifications of bitstreams, streams over the alphabet $\{0,1\}$. Due to their simplicity, bitstreams can be embedded in almost every non-trivial coinductive structure. Specifications of bitstreams are inherently sorted, with a sort $B$ for bits, and a sort $S$ for bitstreams. To this end, we introduce sorted terms. Let $\mathcal{S}$ be a set of sorts; an *$\mathcal{S}$-sorted set* $C$ is a family of sets $\{C_s\}_{s\in\mathcal{S}}$. Let $C$ and $D$ be $\mathcal{S}$-sorted sets. Then an $\mathcal{S}$-sorted *function* (or *map*) from $C$ to $D$ is a function $f : C \to D$ such that $f(C_s) \subseteq D_s$ for all $s \in \mathcal{S}$, that is, a function that respects the sorts.

An $\mathcal{S}$-sorted signature $\Sigma$ is a set of symbols $f \in \Sigma$, each having a type $(s_1, \ldots, s_n, s) \in \mathcal{S}^{n+1}$, denoted by $f :: s_1 \times \ldots \times s_n \to s$, where $n$ is the arity of $f$. Let $\mathcal{X}$ be an $\mathcal{S}$-sorted set of *variables*. The $\mathcal{S}$-sorted set of *terms* $Ter(\Sigma, \mathcal{X})$ is inductively defined by:

- $\mathcal{X}_s \subseteq Ter(\Sigma, \mathcal{X})_s$ for every $s \in \mathcal{S}$, and
- $f(t_1, \ldots, t_n) \in Ter(\Sigma, \mathcal{X})_s$ if $f :: s_1 \times \ldots \times s_n \to s$, $f \in \Sigma$, and $t_1 \in Ter(\Sigma, \mathcal{X})_{s_1}, \ldots, t_n \in Ter(\Sigma, \mathcal{X})_{s_n}$.

An $\mathcal{S}$-sorted *equation* $\ell = r$ consists of terms $\ell, r \in Ter(\Sigma, \mathcal{X})_s \times Ter(\Sigma, \mathcal{X})_s$ for some $s \in \mathcal{S}$.

**Definition 1.** A *bitstream signature* $\Sigma$ is an $\mathcal{S}$-sorted signature with $\mathcal{S} = \{B, S\}$ such that $0, 1, : \in \Sigma$ where $0, 1 :: B$ are called *bits*, and the infix symbol ':' of type $B \times S \to S$ is the *stream constructor*. An *equational bitstream specification* over $\Sigma$ is a *finite* set $E$ of equations over $\Sigma$.

From now on we let $\mathcal{S} = \{B, S\}$.

**Definition 2.** A *stream algebra* $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ consists of:

(i) an $\mathcal{S}$-sorted domain $A$; $A_B = \{0,1\}$ and $\varnothing \neq A_S \subseteq \{0,1\}^{\mathbb{N}}$,

(ii) for every $f :: s_1 \times \ldots \times s_n \to s \in \Sigma$ an *interpretation* $[\![f]\!] : A_{s_1} \times \ldots A_{s_n} \to A_s$ ,
(iii) $: \in \Sigma$ with $[\![:]\!](x, \sigma) = x : \sigma$,
(iv) $0, 1 \in \Sigma$ with $[\![0]\!] = 0$ and $[\![1]\!] = 1$.

The clause (iv) of the definition is optional; in fact, the results in this paper are independent of its presence. We have included it since the models where $[\![0]\!] = [\![1]\!]$ are trivial, in the sense that then all bitstreams are equal.

**Definition 3.** Let $\mathcal{A} = \langle A, [\![\cdot]\!] \rangle$ be a stream algebra. Moreover, let $\alpha : \mathcal{X} \to A$ be a variable assignment. As usual, the *interpretation of terms* $[\![\cdot]\!]^{\mathcal{A}}_{\alpha} : Ter(\Sigma, \mathcal{X}) \to A$ is defined inductively by:

$$[\![x]\!]^{\mathcal{A}}_{\alpha} = \alpha(x) \quad [\![f(t_1, \ldots, t_n)]\!]^{\mathcal{A}}_{\alpha} = [\![f]\!]([\![t_1]\!]^{\mathcal{A}}_{\alpha}, \ldots, [\![t_n]\!]^{\mathcal{A}}_{\alpha})$$

Then $\mathcal{A}$ is called a *(stream) model* of $E$ if $[\![\ell]\!]_{\alpha} = [\![r]\!]_{\alpha}$ for every $\ell = r \in E$ and $\alpha : \mathcal{X} \to A$. We write $[\![\cdot]\!]_{\alpha}$ for $[\![\cdot]\!]^{\mathcal{A}}_{\alpha}$ whenever $\mathcal{A}$ is clear from the context. For ground terms $t \in Ter(\Sigma, \varnothing)$, we have $[\![t]\!]_{\alpha} = [\![t]\!]_{\beta}$ for all assignments $\alpha, \beta$; we then write $[\![t]\!]$ for short.

Thus, we interpret function symbols as functions over bits and bitstreams as imposed by their sort. In particular, terms of type $S$ are interpreted as bitstreams. In contrast to [17], our setup does not allow for *confusion* in the models. Recall that confusion means that the models can contain multiple representatives for the same stream.

**Definition 4.** We say that a model $\mathcal{A} = \langle A, [\![\cdot]\!] \rangle$ is *full* if its domain contains all bitstreams, $A_S = \{0, 1\}^{\mathbb{N}}$.

## 4. Turing Machines as Equational Specifications

We now define a set of standard equations (for bitstream specifications) that will be used throughout this paper:

$$\left.\begin{aligned} \mathsf{zeros} &= 0 : \mathsf{zeros} \qquad \mathsf{ones} = 1 : \mathsf{ones} \\ \mathsf{zip}_1(\tau) &= \tau \\ \mathsf{zip}_2(x : \tau_1, \tau_2) &= x : \mathsf{zip}_2(\tau_2, \tau_1) \\ \mathsf{zip}_n(\tau_1, \ldots, \tau_n) &= \mathsf{zip}_2(\tau_1, \mathsf{zip}_{n-1}(\tau_2, \ldots, \tau_n)) \quad (n > 2) \end{aligned}\right\} \quad (9)$$

To give an example, $\mathsf{zip}_3(\sigma, \tau, \rho) = \sigma(0) : \tau(0) : \sigma(1) : \rho(0) : \sigma(2) : \tau(1) : \sigma(3) : \rho(1) : \sigma(4) : \tau(2) : \ldots$, writing $\sigma(i)$ for the $i$'th entry of the stream $\sigma$.

We emphasize that all systems of equations in this paper are finite. To that end, we extend the specifications only by those equations from (9) that are needed by the specification, that is, the equations $\mathsf{zip}_n(\ldots) = \ldots$ for which a symbol $\mathsf{zip}_m$ with $n \leq m$ occurs in the specification.

**Lemma 1.** *In every stream model $\mathcal{A} = \langle A, [\![\cdot]\!] \rangle$ of a specification including the equations from* (9) *we have:*

*(i)* $[\![\mathsf{zeros}]\!] = 0^{\omega}$ *and* $[\![\mathsf{ones}]\!] = 1^{\omega}$,
*(ii) for all* $\sigma_1, \ldots, \sigma_k \in A_S$, $k \geq 2$ *and* $n \in \mathbb{N}$:
$[\![\mathsf{zip}_1]\!](\sigma_1) = \sigma_1$,
$[\![\mathsf{zip}_k]\!](\sigma_1, \ldots, \sigma_k)(2n) = \sigma_1(n)$
$[\![\mathsf{zip}_k]\!](\sigma_1, \ldots, \sigma_k)(2n+1) = [\![\mathsf{zip}_{k-1}]\!](\sigma_2, \ldots, \sigma_k)(n)$

A *Turing machine* $\mathsf{M}$ is a quadruple $\langle Q, \Gamma, q_0, \delta \rangle$ consisting of a finite set of states $Q$, an initial state $q_0 \in Q$, a finite alphabet $\Gamma$ containing a designated *blank* symbol $\square$, and a partial *transition function* $\delta : Q \times \Gamma \rightharpoonup Q \times \Gamma \times \{L, R\}$.

For convenience, we restrict $\Gamma$ to the alphabet $\Gamma = \{0, 1\}$ where 0 is the blank symbol $\square$, and we denote Turing machines by triples $\langle Q, q_0, \delta \rangle$. As input for the Turing machines we typically use a unary number representation $11 \ldots 1$ ($n$-times) to encode the number $n$. Of course, another encoding is possible, as long as the encoding is computable, and the Turing machine is able to detect the end of the input (since 0 is part of the input alphabet and it is also the blank symbol).

We define a translation of Turing machines to equational specifications of bitstream functions, based on the standard translation to term rewriting systems from [24]. However, we represent the tape using streams instead of finite lists, and have one instead of four rules for 'extending' the tape. In particular, the equation for extending the tape is the equation for $\mathsf{zeros}$ from (9). The terms of the shape $q(\sigma, \tau)$ represent configurations of the Turing machine, where the stream $\tau$ contains the tape content below and right of the head, and $\sigma$ the tape content left of the head. Notably, the head of the machine stands on the first symbol of $\tau$.

**Definition 5.** Let $\mathsf{M} = \langle Q, q_0, \delta \rangle$ be a Turing machine. We define the specification $E_{\mathsf{M}}$ to consist of the following equations:

$$\begin{aligned} q(x, b : y) &= q'(b' : x, y) && \text{for every } \delta(q, b) = \langle q', b', R \rangle \\ q(a : x, b : y) &= q'(x, a : b' : y) && \text{for every } \delta(q, b) = \langle q', b', L \rangle \end{aligned}$$

and for halting configurations additionally:

$$q(x, b : y) = b \qquad \text{whenever } \delta(q, b) \text{ undefined}$$

with the signature $\Sigma = \{0, 1, :\} \cup Q$ with types $q :: S \times S \to B$ for every symbol $q \in Q$, and $0, 1 :: B$ and ':' of type $B \times S \to S$. Moreover, we use $R_{\mathsf{M}}$ to denote the term rewriting system obtained from $E_{\mathsf{M}}$ by orienting all equations from left to right.

Apart from the additional rule for termination, the translation $R_{\mathsf{M}}$ is standard, and the rewrite rules model the transition relation of Turing machines in one-to-one fashion. So we take the liberty to define input of tuples $\langle n_1, \ldots, n_k \rangle \in \mathbb{N}^k$ and oracles directly on the term representations. We pass $k$-tuples $\langle n_1, \ldots, n_k \rangle \in \mathbb{N}^k$ of natural numbers as input to a Turing machine by choosing the following start configuration $q_0(\mathsf{zeros},\ \mathsf{zip}_{k+1}(\underline{k}, \underline{n_1},\ \ldots,\ \underline{n_k}))$ where $\underline{n}$ stands for $(1 :)^n\ \mathsf{zeros}$. The particular encoding of tuples is not crucial, but $\mathsf{zip}_{k+1}(\underline{k}, \underline{n_1},\ \ldots,\ \underline{n_k})$ is for equational specifications more convenient than the Gödel encoding.

We obtain machines with oracles $\xi_1, \ldots, \xi_m \subseteq \mathbb{N}$ by writing the oracles elementwise interleaved on the tape left of the head:

**Notation 1.** *For $n \in \mathbb{N}$ we use $\underline{n}$ to abbreviate $(1 :)^n : \mathsf{zeros}$. For $\xi \subseteq \mathbb{N}$, we let $\underline{\xi}$ denote the stream $\chi_\xi(0) : \chi_\xi(1) : \chi_\xi(2) : \ldots$ where $\chi_\xi$ is the characteristic function of $\xi$. We write $\vec{\alpha}$ short for $\alpha_1, \ldots, \alpha_k$ and $\underline{\vec{\alpha}}$ for $\underline{\alpha_1}, \ldots, \underline{\alpha_k}$ if $k$ is clear from the context.*

*For a term rewriting system $R$, we write $\to_R$ for a rewrite step with respect to $R$, and $\to_R^*$ is the reflexive-transitive closure of $\to_R$.*

**Definition 6.** Let $\mathsf{M} = \langle Q, q_0, \delta \rangle$ be a Turing machine. Then for stream terms $\xi_1, \ldots, \xi_m :: S$ and $n_1, \ldots, n_k :: S$, we define

$$\mathsf{M}(\xi_1, \ldots, \xi_m; n_1, \ldots, n_k) := q_0(\mathsf{zip}_m(\xi_1, \ldots, \xi_m),\ \mathsf{zip}_{k+1}(\underline{k}, n_1,\ \ldots,\ n_k))$$

**Definition 7.** A Turing machine $\mathsf{M} = \langle Q, q_0, \delta \rangle$ *halts (with output $b$)* on inputs $n_1, \ldots, n_k \in \mathbb{N}$ with oracles $\xi_1, \ldots, \xi_m \subseteq \mathbb{N}$ if there is a rewrite sequence $\mathsf{M}(\underline{\xi_1}, \ldots, \underline{\xi_m}; \underline{n_1},\ \ldots,\ \underline{n_k}) \to_{R_{\mathsf{M}}}^* b$, where $b \in \{0, 1\}$. Here $\underline{\xi}$ is short for the stream $\chi_\xi(0) : \chi_\xi(1) : \chi_\xi(2) : \ldots$ where $\chi_\xi$ is the characteristic function of $\xi$.

Note that the initial term is infinite due to the oracles, nevertheless we consider only finite reduction sequences. Due to the rules for $\mathsf{zip}_n$ and $\mathsf{zeros}$, there are infinite rewrite sequences even if the Turing machine halts. However, $R_{\mathsf{M}}$ is orthogonal and therefore outermost-fair rewriting (or lazy evaluation) is normalizing, that is, computes the (unique) normal form $b \in \{0, 1\}$ if it exists.

**Definition 8.** A *$k$-ary predicate $P$ with $m$ oracles* is a relation $P \subseteq \wp(\mathbb{N})^m \times \mathbb{N}^k$. Then $P$ is called *decidable* if there is a Turing

machine $\mathsf{M}$ such that for all $\vec{\xi} \in \wp(\mathbb{N})^m$ and $\vec{n} \in \mathbb{N}^k$: $\mathsf{M}$ halts on input $\vec{n}$ with oracles $\vec{\xi}$, and the output is 1 if and only if $P(\vec{\xi}, \vec{n})$.

In correspondence with Definition 6 we define for $\xi_1, \ldots, \xi_m$, $n_1, \ldots, n_k \in \{0,1\}^\omega$, $[\![\mathsf{M}]\!](\xi_1, \ldots, \xi_m; n_1, \ldots, n_k)$ as shorthand for $[\![q_0]\!]([\![\mathsf{zip}_m]\!](\xi_1, \ldots, \xi_m),\ [\![\mathsf{zip}_{k+1}]\!]([\![\underline{k}]\!], n_1,\ \ldots,\ n_k))$. Then for the models of Turing machine specifications we have:

**Lemma 2.** *Let $P \subseteq \wp(\mathbb{N})^m \times \mathbb{N}^k$ be decidable, and $\mathsf{M} = \langle Q, q_0, \delta\rangle$ the corresponding Turing machine. Then in every stream model $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ of a specification including the equations from* (9) *and $E_\mathsf{M}$ we have for every $\vec{\xi} \in \wp(\mathbb{N})^m$ and $\vec{n} \in \mathbb{N}^k$: $(\vec{\xi}, \vec{n}) \in P$ if and only if $[\![\mathsf{M}]\!](\xi_1, \ldots, \xi_m; n_1,\ \ldots,\ n_k) = 1$.*

*Proof.* $P$ is decidable, hence $\mathsf{M}(\underline{\xi_1}, \ldots, \underline{\xi_m}; \underline{n_1},\ \ldots,\ \underline{n_k})$ has a nf in $\{0,1\}$, and the normal form is 1 if and only if $(\vec{\xi}, \vec{n}) \in P$. □

## 5. Levels of Undecidability

We briefly introduce complexity related notions that are relevant for this paper: promise problems, reducibility, hardness and completeness, and the arithmetical and the analytical hierarchy. For more details, we refer to the standard textbooks [18, 22].

**Definition 9.** Let $A \subseteq P \subseteq \mathbb{N}$. The *promise (membership) problem for $A$ with promise $P$* is the question of deciding on the input of $n \in P$ whether $n \in A$. For the case $P = \mathbb{N}$, we speak of the *membership problem for $A$*.

We identify the membership problem for $A$ with the set $A$ itself, and the promise problem for $A$ with promise $P$ with the pair $\langle A, P\rangle$, also denoted by $A|_P$.

**Definition 10.** Let $A, B, P, Q \subseteq \mathbb{N}$. Then $A|_P$ *can be (many-one) reduced to* $B|_Q$, denoted $A \leq B$, if there exists a partial recursive function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ such that $P \subseteq \mathit{domain}(f)$, $f(P) \subseteq Q$, and $\forall n \in P.\, n \in A \Leftrightarrow f(n) \in B$.

**Definition 11.** Let $B, Q \subseteq \mathbb{N}$ and $\mathcal{P} \subseteq \wp(\mathbb{N}) \times \wp(\mathbb{N})$. Then $B|_Q$ is called *$\mathcal{P}$-hard* if every $A|_P \in \mathcal{P}$ can be reduced to $B|_Q$. Moreover, $B|_Q$ is *$\mathcal{P}$-complete* if additionally $B|_Q$ can be reduced to some $A|_P \in \mathcal{P}$.

We stress that Definition 11 does not require that a $\mathcal{P}$-complete promise problem $B|_Q$ is member of $\mathcal{P}$ itself. This allows for classifying promise problem using the usual arithmetic and analytical hierarchy (for membership problems).

**Lemma 3.** *If $A|_P$ can be reduced to $B|_Q$ and $A|_P$ is* $\mathcal{P}$-hard*, then $B$ is* $\mathcal{P}$-hard*.*

We use $\langle\!\langle\cdot\rangle\!\rangle$ to denote the well-known Gödel encoding of finite *lists* of numbers as elements of $\mathbb{N}$: $\langle\!\langle n_1, \ldots, n_k\rangle\!\rangle := p_1^{n_1+1} \cdot \ldots \cdot p_k^{n_k+1}$, where $p_1 < p_2 < \ldots < p_k$ are the first $k$ prime numbers.

We define the arithmetical and analytical hierarchies:

**Definition 12.** Let $\Sigma^0_0 := \Pi^0_0 := \Delta^0_0$ be the collection of recursive sets of natural numbers (the decidable problems). Then for $n \geq 1$, we define:

- $\Sigma^0_n$ consists of sets $\{n \mid \exists x \in \mathbb{N}.\, \langle\!\langle x, n\rangle\!\rangle \in B\}$ with $B \in \Pi^0_{n-1}$,
- $\Pi^0_n$ consists of sets $\{n \mid \forall x \in \mathbb{N}.\, \langle\!\langle x, n\rangle\!\rangle \in B\}$ with $B \in \Sigma^0_{n-1}$,
- $\Delta^0_n := \Sigma^0_n \cap \Pi^0_n$.

The *arithmetical hierarchy* consists of the classes $\Pi^0_n$, $\Sigma^0_n$ and $\Delta^0_n$ for $n \in \mathbb{N}$.

For example, the membership $a \in A$ for every set $A \in \Pi^0_2$ can be defined by a formula of the form $\forall x_1.\exists x_2.\forall x_3.P(a, x_1, x_2, x_3)$ where $P$ is a decidable predicate.

The analytical hierarchy extends this classification of sets to formulas of the language of second-order arithmetic, that is, with set (or equivalently function) quantifiers. The following definition makes use of a result from recursion theory, see [18], stating that if there is at least one set quantifier, then two number quantifiers suffice (for functions quantifiers, one number quantifier suffices).

**Definition 13.** Let $\Sigma^1_0 := \Pi^1_0 := \Delta^1_0 = \bigcup_{n\in\mathbb{N}} \Pi^0_n$ be the set of all arithmetic predicates. A set $A \subseteq \mathbb{N}$ is in $\Pi^1_n$ for $n > 0$ if there is a decidable predicate $P$ with $m$ oracles such that for all $a \in \mathbb{N}$:

$$a \in A \iff \forall\xi_1.\, \exists\xi_2. \ldots \exists\xi_m.\, \forall x_1.\, \exists x_2.\, P(\xi_1, \ldots, \xi_n, a, x_1, x_2)$$
$$a \in A \iff \forall\xi_1.\, \exists\xi_2. \ldots \forall\xi_m.\, \exists x_1.\, \forall x_2.\, P(\xi_1, \ldots, \xi_n, a, x_1, x_2)$$

for $n$ even, and $n$ odd, respectively. Here, $\xi_1, \ldots, \xi_m \subseteq \mathbb{N}$, the corresponding quantifiers are set quantifiers, and $x_1, x_2 \in \mathbb{N}$ with number quantifiers. Then $A$ is in $\Sigma^1_n$, if the condition holds with all $\forall$ and $\exists$ quantifiers swapped. Finally, $\Delta^1_n = \Pi^1_n \cap \Sigma^1_n$.

## 6. Equality in Models

In this section we study the complexity of different model-theoretic semantics of equivalence of bitstream specifications. Based on the notion of models for bitstream specifications from Section 3, we first formalize the equivalences that we consider.

For all of the following model-theoretic equivalences, we have the choice whether or not we require the models to be full, that is, their domain contains all bitstreams. For example, we can consider the equality of terms in all models or in all full models:

**Definition 14.** Let $E$ be a bitstream specification over $\Sigma$, and $s, t \in \mathit{Ter}(\Sigma, \mathcal{X})$ with $s, t :: S$. Then $s$ and $t$ are said to be

- *equal in all models of $E$* if
  $$\mathcal{A} \models E \text{ implies } \mathcal{A} \models s = t \text{ for all stream algebras } \mathcal{A},$$
- *equal in all full models of $E$* if
  $$\mathcal{A} \models E \text{ implies } \mathcal{A} \models s = t \text{ for all full stream algebras } \mathcal{A}.$$

The set of solutions of a term $s$ in a specification $E$ is the set of interpretations $[\![s]\!]$ of $s$ in all models satisfying $E$:

**Definition 15.** Let $E$ be a bitstream specification over $\Sigma$, and $s \in \mathit{Ter}(\Sigma, \varnothing)$ with $s :: S$. Then the set of

- *solutions of $s$ in $E$ with respect to all models* is
  $$[\![s]\!]_E = \{\, [\![s]\!]^\mathcal{A} \mid \mathcal{A} \models E \,\},$$
- *solutions of $s$ in $E$ with respect to all full models* is
  $$[\![s]\!]_{E,\,\text{full}} = \{\, [\![s]\!]^\mathcal{A} \mid \mathcal{A} \text{ full}, \mathcal{A} \models E \,\}.$$

Here it suffices to consider only ground terms $s \in \mathit{Ter}(\Sigma, \varnothing)$. For terms $t \in \mathit{Ter}(\Sigma, \mathcal{X})$ with variables, the set of solutions can be defined as $[\![t]\!]_E = \{\, [\![t]\!]^\mathcal{A}_\alpha \mid \mathcal{A} \models E, \alpha : \mathcal{X} \to A \,\}$. However, then $[\![t]\!]_E = [\![s]\!]_E$ if $s$ is the ground term obtained from $t$ by interpreting the variables in $t$ as fresh constants (formally, this amounts to an extension of the signature).

**Definition 16.** Let $E_s$ and $E_t$ be bitstream specifications over $\Sigma_s$ and $\Sigma_t$, respectively. Let $s \in \mathit{Ter}(\Sigma_s, \varnothing)$ and $t \in \mathit{Ter}(\Sigma_t, \varnothing)$. Then $s$ and $t$ have

- *equal solutions over all models* if $[\![s]\!]_{E_s} = [\![t]\!]_{E_t}$,
- *equal solutions over all full models* if $[\![s]\!]_{E_s,\,\text{full}} = [\![t]\!]_{E_t,\,\text{full}}$.

**Definition 17.** Let $E$ be bitstream specifications over $\Sigma$, and $s \in \mathit{Ter}(\Sigma, \varnothing)$. Then $s$ is said to have

- *a unique solution over all models* if $|[\![s]\!]_E| = 1$,
- *a unique solution over all full models* if $|[\![s]\!]_{E,\,\text{full}}| = 1$,
- *a solution over all models* if $|[\![s]\!]_E| \geq 1$,
- *a solution over all full models* if $|[\![s]\!]_{E,\,\text{full}}| \geq 1$,
- *at most one solution over all models* if $|[\![s]\!]_E| \leq 1$,

– *at most one solution over all full models* if $|[\![s]\!]_{E,\,\mathrm{full}}| \leq 1$.

### 6.1 Auxiliary Definitions

First, we define a few (systems of) equations that are repeatedly used throughout this section. The following function $\mathsf{is_{zeros}}$ that maps $\mathsf{zeros}$ to $\mathsf{ones}$ and every other bitstreams to $\mathsf{zeros}$:

$$\left.\begin{array}{ll} \mathsf{is_{zeros}}(\mathsf{zeros}) = \mathsf{ones} & \mathsf{is_{zeros}}(0:\sigma) = \mathsf{is_{zeros}}(\sigma) \\ & \mathsf{is_{zeros}}(1:\sigma) = \mathsf{zeros} \end{array}\right\} \quad (10)$$

This function does exactly what its name suggests; it checks whether the argument is the stream of zeros. We use the bit 0 or the stream $\mathsf{zeros}$ for *false*, and 1 and $\mathsf{ones}$ for *true*.

We focus on specifications of bitstreams, and encode streams of natural numbers as bitstreams via the sequence of run-length of ones. For instance, the stream $3:1:0:2:\ldots$ is encoded as $1:1:1:0:1:0:0:1:1:0:\ldots$. We then define functions $\mathsf{uhd}$ and $\mathsf{utl}$ that are the unary counterpart for head and tail on streams of natural numbers:

$$\left.\begin{array}{ll} \mathsf{uhd}(0:\sigma) = \mathsf{zeros} & \mathsf{utl}(0:\sigma) = \sigma \\ \mathsf{uhd}(1:\sigma) = 1:\mathsf{uhd}(\sigma) & \mathsf{utl}(1:\sigma) = \mathsf{utl}(\sigma) \end{array}\right\} \quad (11)$$

For instance, we have

$$\mathsf{uhd}(1:1:1:0:1:0:0:1:1:\ldots) = 1:1:1:\mathsf{zeros}$$
$$\mathsf{utl}(1:1:1:0:1:0:0:1:1:\ldots) = 1:0:0:1:1:\ldots$$

The following lemma summarizes these properties:

**Lemma 4.** *In every stream model $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ of a specification including the equations from* (10) *and* (11) *we have:*

*(i)* $[\![\mathsf{is_{zeros}}]\!](0^\omega) = 1^\omega$,
$[\![\mathsf{is_{zeros}}]\!](w) = 0^\omega$ *for every* $w \in A_S \setminus \{0^\omega\}$,
*(ii)* $[\![\mathsf{uhd}]\!](1^n\,0\,w) = 1^n\,0^\omega$ *for every* $w \in A_S$,
$[\![\mathsf{uhd}]\!](1^\omega) = 1^\omega$,
*(iii)* $[\![\mathsf{utl}]\!](1^n\,0\,w) = w$ *for every* $w \in A_S$.

Note that all interpretations are uniquely defined, apart from the combination $[\![\mathsf{utl}]\!](1^\omega)$ which can be any stream depending on the model. To avoid this case, we need means to ensure that a certain bitstream is a valid encoding of a stream of natural numbers, that is, the stream contains infinitely many zeros:

$$\left.\begin{array}{ll} \mathsf{natstr}(\mathsf{ones}) = \mathsf{zeros} & \mathsf{natstr}(0:\sigma) = 1:\mathsf{natstr}(\sigma) \\ & \mathsf{natstr}(1:\sigma) = \mathsf{natstr}(\sigma) \end{array}\right\} \quad (12)$$

Then an equation $\mathsf{natstr}(X) = \mathsf{ones}$ guarantees that $[\![X]\!]$ represents a stream of natural numbers:

**Lemma 5.** *In every stream model $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ of a specification including the equations from* (12) *we have:* $[\![\mathsf{natstr}]\!](w) = 1^\omega$ *if and only if $w$ contains infinitely many zeros.*

*Proof.* The equations on the right 'walk' over the stream, deleting 1's and converting 0's to 1's. If the stream contains infinitely many 0's, then an infinite stream of 1's will be produced. However, if some tail of the stream contains only 1's then the equation on the left ensures that the interpretation is unequal to $1^\omega$. □

**Definition 18.** Let $\mathsf{M} = \langle Q, q_0, \delta\rangle$ be a Turing machine. Then the *canonical model* $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ for the union of the specifications $E_\mathsf{M}$, (9), (10), (11) and (12) consists of the domain $A_S = \{0,1\}^\mathbb{N}$ with interpretations $[\![\cdot]\!]$ as given in Lemmas 4 and 5, extended by

(i) $[\![\mathsf{utl}]\!](1^\omega) = 1^\omega$,
(ii) for every $\xi_1, \ldots, \xi_m, n_1, \ldots, n_k \subseteq \mathbb{N}$:
$[\![q]\!](\vec{\xi}, \vec{n}) = 1$ whenever $[\![q]\!](\vec{\underline{\xi}}, \vec{\underline{n}}) \to^* 1$, and
$[\![q]\!](\vec{\xi}, \vec{n}) = 0$ otherwise.

**Lemma 6.** *The canonical model is a model of the union of the equational specifications $R_\mathsf{M}$,* (9)*,* (10)*,* (11) *and* (12)*.*

*Proof.* The rewrite system $R_\mathsf{M}$ is orthogonal, consequently we have finitary confluence and infinitary unique normal forms [24]. Hence, we can employ a normal forms semantics for $[\![q]\!]$ (where we map terms without normal forms to 0). For the remaining equations, it is easy to see that the chosen semantics forms a model. □

### 6.2 Equality in all Models

For the complexity of equality in all models we obtain:

**Theorem 1.** *The following problem is $\Pi^1_1$-complete:*
INPUT: *Bitstream specification $E$, terms $s, t :: S$.*
QUESTION: *Are $s$ and $t$ equal in all models of $E$?*

*Proof.* The well-foundedness problem for decidable binary relations is known to be $\Pi^1_1$-complete, that is, the problem of deciding on the input of a decidable binary predicate $M \subseteq \mathbb{N} \times \mathbb{N}$ (given in the form of a Turing machine), whether $M$ is well-founded. We reduce this problem to an equality problem. Let $M \subseteq \mathbb{N} \times \mathbb{N}$ be a decidable predicate, and $\mathsf{M} = \langle Q, q_0, \delta\rangle$ the corresponding Turing machine. We define the following specification $E$:

$$\mathsf{S} = \mathsf{is_{zeros}}(\mathsf{run}(1,\mathsf{X})) \qquad \mathsf{natstr}(X) = \mathsf{ones}$$
$$\mathsf{run}(0,\sigma) = \mathsf{ones}$$
$$\mathsf{run}(1,\sigma) = 0 : \mathsf{run}(\overbrace{\mathsf{M}(\mathsf{zeros};\mathsf{uhd}(\sigma),\mathsf{uhd}(\mathsf{utl}(\sigma)))}^{\Phi(\sigma)}, \mathsf{utl}(\sigma))$$

together with the equations from $E_\mathsf{M}$ and (9), (10), (11) and (12). We prove that: $E \models \mathsf{S} = \mathsf{zeros}$ if and only if $M$ is well-founded.

For '$\Rightarrow$' let $M$ be non-well-founded, and $n_0\ M\ n_1\ M\ n_2\ M \ldots$ be an infinite chain. We construct a $\Sigma$-algebra $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ such that $\mathcal{A} \models E$ but not $\mathcal{A} \models \mathsf{S} = \mathsf{zeros}$. We define $\mathcal{A}$ as an extension of the canonical model (Definition 18). The values of $[\![\Phi(\sigma)]\!]$ and $[\![\mathsf{utl}(\sigma)]\!]$ are determined by the canonical model, and together with the equations for $\mathsf{run}$ we obtain for every stream $\xi \in \{0,1\}^\omega$: $[\![\mathsf{run}]\!](0,\xi) = 1^\omega$, and $[\![\mathsf{run}]\!](1,\xi) = 0 : [\![\mathsf{run}]\!]([\![\Phi(\underline{\xi})]\!], [\![\mathsf{utl}]\!](\xi))$. Hence, there is a unique interpretation $[\![\mathsf{run}]\!]$ that results in a model for the equations of $\mathsf{run}$. We define $\kappa_i = 1^{n_i}\,0\,1^{n_{i+1}}\,0\,1^{n_{i+2}}\ldots$ and we let $\underline{n} = 1^n\,0^\omega$. Then for $i \in \mathbb{N}$ we have

$$\begin{aligned}[\![\mathsf{run}]\!](1,\kappa_i) &= 0 : [\![\mathsf{run}]\!]([\![\Phi]\!](\kappa_i), \kappa_{i+1}) \\ &= 0 : [\![\mathsf{run}]\!]([\![\mathsf{M}]\!](\underline{n_i}, \underline{n_{i+1}}), \kappa_{i+1}) = 0 : [\![\mathsf{run}]\!](1, \kappa_{i+1})\end{aligned}$$

since we have that $[\![\mathsf{uhd}]\!](\kappa_j) = \underline{n_j}$ and $[\![\mathsf{utl}]\!](\kappa_j) = \kappa_{j+1}$ for all $j \in \mathbb{N}$ by Lemma 4. Thus, $[\![\mathsf{run}]\!](1,\kappa_0) = 0^\omega$. Let $[\![\mathsf{X}]\!] = \kappa_0$ and $[\![\mathsf{S}]\!] = 1^\omega$. Then $[\![\mathsf{natstr}]\!]([\![\mathsf{X}]\!]) = [\![\mathsf{ones}]\!]$ by Lemma 5, and $[\![\mathsf{S}]\!] = [\![\mathsf{is_{zeros}}]\!]([\![\mathsf{run}]\!](1,[\![\mathsf{X}]\!]))$ by Lemma 4. We have constructed a model, where $[\![\mathsf{S}]\!] = 1^\omega$, and, hence, $E \not\models \mathsf{S} = \mathsf{zeros}$.

For '$\Leftarrow$' let $M$ be well-founded. Let $\mathcal{A}$ be a $\Sigma$-algebra such that $\mathcal{A} \models E$. We show that $[\![\mathsf{S}]\!] = 0^\omega$. Since $[\![\mathsf{natstr}]\!]([\![\mathsf{X}]\!]) = [\![\mathsf{ones}]\!]$, $[\![\mathsf{X}]\!]$ contains infinitely many zeros by Lemma 5. Thus, $[\![\mathsf{X}]\!] = 1^{n_0}\,0\,1^{n_1}\,0\,1^{n_2}\ldots$ for some $n_0,\ n_1, n_2, \ldots \in \mathbb{N}$. Let $\kappa_i = 1^{n_i}\,0\,1^{n_{i+1}}\,0\,1^{n_{i+2}}\ldots$ for $i \in \mathbb{N}$. Then

$$[\![\mathsf{run}]\!](1,\kappa_i) = 0 : [\![\mathsf{run}]\!]([\![\mathsf{M}]\!](\underline{n_i}, \underline{n_{i+1}}), \kappa_{i+1})$$
$$= \begin{cases} [\![\mathsf{run}]\!](1,\kappa_{i+1}) & \text{if } [\![\mathsf{M}]\!](\underline{n_i}, \underline{n_{i+1}}) = 1 \\ [\![\mathsf{run}]\!](0,\kappa_{i+1}) = 1^\omega & \text{if } [\![\mathsf{M}]\!](\underline{n_i}, \underline{n_{i+1}}) = 0 \end{cases}$$

Hence, $[\![\mathsf{run}]\!](1,[\![\mathsf{X}]\!]) = 0^\omega$ if and only if $[\![\mathsf{M}]\!](\underline{n_i}, n_{i+1}) = 1$ for all $i \in \mathbb{N}$. However, this would contradict well-foundedness of $M$. As a consequence, we obtain that $[\![\mathsf{run}]\!](1,[\![\mathsf{X}]\!]) \neq 0^\omega$ and $[\![\mathsf{S}]\!] = \mathsf{is_{zeros}}([\![\mathsf{run}]\!](1,[\![\mathsf{X}]\!])) = 0^\omega$ by Lemma 4. This concludes the $\Pi^1_1$-hardness proof.

To show $\Pi^1_1$-membership, we resort to the Löwenheim–Skolem theorem. It states that if a formula of first-order predicate logic has an uncountable model, then it also has a countable model. Here,

we employ that the domain $A_S$ can be encoded as an arbitrary set with functions $[\![\mathsf{head}]\!] :: A_S \to \{0,1\}$ and $[\![\mathsf{tail}]\!] :: A_S \to A_S$ together with a first-order predicate logic formula that excludes confusion, that is, elements $a, b \in A_S$ with $[\![\mathsf{head}]\!][\![\mathsf{tail}]\!]^n(a) = [\![\mathsf{head}]\!][\![\mathsf{tail}]\!]^n(b)$ for all $n \in \mathbb{N}$ are required to be equal, that is, $a = b$. Likewise, the interpretations of the symbols in $\Sigma$ can be translated to first-order predicates, and validity of the equations to first-order formulas. As a consequence, $\mathcal{A} \models E \wedge [\![s]\!] \neq [\![t]\!]$ can be expressed as first-order formula, and if it has a model, then also a countable one. Hence, it suffices in $\forall \mathcal{A}.\ \mathcal{A} \models E \Rightarrow [\![s]\!] = [\![t]\!]$ to quantify over countable models. For this purpose of quantifying over countable models, a set quantifier $\forall \mathcal{A} \subseteq \mathbb{N}$ suffices. This proves $\Pi^1_1$-membership. □

The following three results are obtained by slight adaptations of the proof of Theorem 1.

**Theorem 2.** *The following problems:*

INPUT: *Bitstream specification $E$, ground term $s :: S$.*
QUESTION: *Does $s$ have (i) at most one solution, (ii) a solution, and (iii) a unique solution over all models of $E$?*

*are (i) $\Pi^1_1$-complete, (ii) $\Sigma^1_1$-complete, and (iii) $\Pi^1_1$-hard, $\Sigma^1_1$-hard and strictly contained in $\Delta^1_2$.*

### 6.3 Equality in all Full Models

In Section 6.2 we have considered models whose domain was any non-empty set of bitstreams ($A_S \subseteq \{0,1\}^\omega$). However, when writing equations such as $\mathsf{even}(x : y : \tau) = x : \mathsf{even}(\tau)$, the intended semantics is often that these equations should hold for all streams, that is, in full models with domain $A_S = \{0,1\}^\omega$. We find that the restriction to full models results in a huge jump of the complexity, which then subsumes the entire analytical hierarchy.

To prepare for the proof, we introduce some auxiliary specifications. We define $\mathsf{nat}$ such that an equation $\mathsf{nat}(X) = \mathsf{ones}$ guarantees that the interpretation $[\![X]\!]$ represents a natural number in unary encoding, that is, $[\![X]\!] = 1^n\,0^\omega$ for $n \in \mathbb{N}$, as follows:

$$\left.\begin{array}{ll} \mathsf{nat}(0:1:\sigma) = \mathsf{zeros} & \mathsf{nat}(1:\sigma) = \mathsf{nat}(\sigma) \\ \mathsf{nat}(0:0:\sigma) = \mathsf{nat}(0:\sigma) & \mathsf{nat}(\mathsf{ones}) = \mathsf{zeros} \end{array}\right\} \quad (13)$$

**Lemma 7.** *In every stream model $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ of a specification including the equations from* (13) *we have: if $[\![\mathsf{nat}]\!](w) = 1^\omega$ then $w = 1^n\,0^\omega$ for some $n \in \mathbb{N}$.*

*Proof.* If a stream is not of the format $1^n\,0^\omega$ for some $n \in \mathbb{N}$ then it is $1^\omega$ or contains $\ldots 01 \ldots$. The last equation rules out the case $1^\omega$ (ensures that the interpretation is not $1^\omega$).

The first three equations are exhaustive in the sense that every stream can be matched by one of them. The first equation rules out streams that contain a 1 after a 0, and the equations two and three 'walk' step by step over the stream (proceed with the tail). □

We moreover define a function $\mathsf{leq}$ such that $\mathsf{leq}(X, Y) = \mathsf{ones}$ guarantees that pointwise $[\![X]\!] \leq [\![Y]\!]$:

$$\left.\begin{array}{l} \mathsf{leq}(0:\sigma, x:\tau) = \mathsf{leq}(\sigma,\tau) \\ \mathsf{leq}(1:\sigma, 1:\tau) = \mathsf{leq}(\sigma,\tau) \\ \mathsf{leq}(1:\sigma, 0:\tau) = \mathsf{zeros} \end{array}\right\} \quad (14)$$

**Lemma 8.** *In every stream model $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ of a specification including the equations from* (14) *we have that if $[\![\mathsf{leq}]\!](\sigma,\tau) = 1^\omega$, then $\sigma$ is pointwise $\leq$ than $\tau$ (for all $\sigma, \tau \in A_S$).*

Lemmas 7 and 8 are valid for non-full models as well. As explained in the introduction, the assumption of full models is crucial to guarantee that equations with variables have to hold for all streams (assigned to the variables) and not only the streams in the model.

**Theorem 3.** *The following problem subsumes the analytical hierarchy:*

INPUT: *Bitstream specification $E$, terms $s, t :: S$.*
QUESTION: *Are $s$ and $t$ equal in all* full *models of $E$?*

The idea of the proof is as follows. We translate formulas of the analytical hierarchy into stream specifications by representing $\forall$ set quantifiers by equations with variables. This simulates a quantification over all streams as the models are *full*, and the equations have to hold for all assignments of the variables.

The $\exists$ set quantifiers are eliminated in favor of Skolem functions $f$, that is, axioms of the form $\forall \vec{x}.\ \exists y.\ \psi(x_1, \ldots, x_n, y)$ are replaced by $\forall \vec{x}.\ \psi(x_1, x_2, \ldots, x_n, f(x_1, \ldots, x_n))$. The interpretation of these functions is determined by the model, and the question whether there exists a model corresponds to an existential quantification over all Skolem functions.

*Proof.* For every analytical set $A$, we reduce the membership problem in $A$ to an equality problem. Every set $A$ of the analytical hierarchy can be defined by

$$a \notin A \iff \forall \xi_1.\ \exists \xi_2.\ \forall \xi_3.\ \ldots \exists \xi_n.\ \forall x_1.\ \exists x_2.\ M(\xi_1, \ldots, \xi_n, a, x_1, x_2) \quad (15)$$

where $n \in \mathbb{N}$ is even (without loss of generality since $\Pi^1_n \subset \Pi^1_{n+1}$) and $M$ a decidable predicate. Let $\mathsf{M} = \langle Q, q_0, \delta\rangle$ the Turing machine corresponding to $M$. Let $a \in \mathbb{N}$ be given. We define $E$ to be the following system of equations:

$$\begin{array}{l} \mathsf{S}(\tau_1, \tau_3, \ldots, \tau_{n-1}) = \mathsf{run}(1,\ \mathsf{zip}_n(\tau_1, \mathsf{g}_2(\tau_1), \tau_3, \mathsf{g}_4(\tau_1,\tau_3), \\ \qquad\qquad \ldots, \tau_{n-1}, \mathsf{g}_n(\tau_1, \tau_3, \ldots, \tau_{n-1})),\ \mathsf{zeros}) \\ \mathsf{S}(\tau_1, \tau_3, \ldots, \tau_{n-1}) = \mathsf{zeros} \\ \mathsf{run}(0, \tau, \gamma_1) = \mathsf{ones} \\ \mathsf{run}(1, \tau, \gamma_1) = 0 : \mathsf{run}(\mathsf{M}(\tau; A, \gamma_1, \mathsf{h}_2(\tau,\gamma_1)),\ \tau,\ 1:\gamma_1) \\ A = (1:)^a\ \mathsf{zeros} \\ \mathsf{nat}(\mathsf{h}_2(\tau,\gamma_1)) = \mathsf{ones} \end{array}$$

together with the equations from $E_{\mathsf{M}}$, (9), and (13). The symbols $g_{2i}$ are typed $S^i \to S$. We claim: $E \models \mathsf{zeros} = \mathsf{ones}$ if and only if $a \in A$. For this purpose it suffices to show that the specification has a model ($\exists \mathcal{A}.\ \mathcal{A} \models E$) if and only if the formula in the right-hand side of (15) is valid.

The idea is that the specification models a Skolem normal form of the analytical formula in (15). The $\forall$ set quantifiers are modeled by an equation with stream variables; recall that equations have to hold for all assignments of the variables. In particular, the variables $\tau_1, \tau_3, \ldots, \tau_{n-1}$ in the first equation $\mathsf{S}(\tau_1, \tau_3, \ldots, \tau_{n-1}) = \ldots$ model the set quantifiers $\forall \xi_1, \ldots, \forall \xi_{n-1}$, respectively. The $\exists$ set quantifiers are modeled by Skolem functions $g_2, g_4, \ldots, g_n$ which in the specification are stream functions that get the value of the preceding $\forall$ quantifiers as arguments. These stream functions $g_{2i}$ are unspecified and can be 'freely chosen' by the model $\mathcal{A}$. Thus, the existential quantification over the Skolem functions corresponds to the existential quantification over all models in $\exists \mathcal{A}.\ \mathcal{A} \models E$.

The streams $\tau_1, g_2(\tau_i), \ldots, \tau_{n-1}, g_n(\tau_1, \tau_3, \ldots, \tau_{n-1})$ that represent the values of the set quantifiers are then interleaved by $\mathsf{zip}_n$, and passed as the second argument, named $\tau$, to $\mathsf{run}$; this argument serves as the left side of the tape for every invocation of the Turing machine $\mathsf{M}$.

The $\forall x_1$ number quantifier is modeled by the third argument $\gamma_1$ of $\mathsf{run}$. The initial value of $\gamma_1$ is $\mathsf{zeros}$, and '$1 : \square$' is prepended (corresponding to counting up) each time the Turing machine halts with output 1. The number quantifier $\exists x_2$ is modeled by the Skolem function $\mathsf{h}_2$ for which the equation $\mathsf{nat}(\mathsf{h}_2(\tau,\gamma_1)) = \mathsf{ones}$ ensures

by Lemma 4 that the interpretation $[\![\mathsf{h}_2(\tau,\gamma_1)]\!]$ is a unary encoding of a natural number. Then the term $\mathsf{M}(\tau; A, \gamma_1, \mathsf{h}_2(\tau,\gamma_1))$ with $\tau = \mathsf{zip}_n(\tau_1, \mathsf{g}_2(\tau_1), \tau_3, \mathsf{g}_4(\tau_1,\tau_3), \ldots, \tau_{n-1}, \mathsf{g}_n(\tau_1,\tau_3,\ldots,\tau_{n-1}))$ corresponds precisely to $M(\xi_1,\ldots,\xi_n,a,x_1,x_2)$ in (15).

For '$\Leftarrow$', assume that the formula in (15) is valid. We construct a model $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ as an extension of the canonical model (Definition 18). For $[\![g_2]\!], [\![g_4]\!], \ldots, [\![g_n]\!], [\![h_2]\!]$ we pick the Skolem functions for the quantifiers $\exists\xi_2, \exists\xi_4, \ldots, \exists\xi_n, \exists x_2$, respectively (where $[\![h_2]\!]$ is a stream function that works on the unary encoding of natural numbers). For $\sigma \in \{0,1\}^\omega$, we define $[\![\mathsf{nat}]\!](\sigma) = 1^\omega$ if $\sigma$ is of the form $1^n\,0^\omega$, and $0^\omega$, otherwise. The definition of $[\![\mathsf{run}]\!]$ is analogous to the proof of Theorem 1. Finally, we define $[\![\mathsf{S}]\!](\tau_1,\tau_2,\ldots,\tau_{n-1}) = 0^\omega$ for all $\tau_1,\tau_2,\ldots,\tau_{n-1} \in \{0,1\}^\omega$, and $[\![A]\!] = 1^a\,0^\omega$. Then it is straightforward to verify that $\mathcal{A}$ is a model of the specification.

For '$\Rightarrow$', let $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ be a model of the specification. Then we let the existential quantifiers $\exists\xi_2, \exists\xi_4, \ldots, \exists\xi_n$ and $\exists x_2$ in (15) behave according to the interpretations $[\![g_2]\!], [\![g_4]\!], \ldots, [\![g_n]\!], [\![h_2]\!]$, respectively (here the translation from sets $\xi \subseteq \mathbb{N}$ to streams $\underline{\xi}$ is as usual). Assume that there exists an assignment of the $\forall$ quantifiers $\forall\xi_1, \forall\xi_2, \ldots, \forall\xi_{n-1}$ and $\forall x_2$ for which the formula in (15) is not valid, that is, $M(\xi_1,\ldots,\xi_n,a,x_1,x_2)$ does not hold where the existential choices are governed by the model as described above. We translate this 'counterexample' back to the model by considering $[\![\mathsf{S}]\!](\underline{\xi_1}, \underline{\xi_3} \ldots, \underline{\xi_{n-1}})$. As in the proof of Theorem 1, it is then straightforward to show that $[\![\mathsf{S}]\!](\underline{\xi_1}, \underline{\xi_3} \ldots, \underline{\xi_{n-1}}) \neq 0^\omega$. However, this contradicts the assumption of $\mathcal{A}$ being a model due to the equation $\mathsf{S}(\tau_1,\tau_3,\ldots,\tau_{n-1}) = \mathsf{zeros}$. □

The proof of Theorem 3 immediately yields the following:

**Theorem 4.** *Each of the following problems (i), (ii), and (iii), subsume the analytical hierarchy:*

INPUT: *Bitstream specification $E$, ground term $t :: S$.*
QUESTION: *Does $t$ have: (i) a solution, (ii) a unique solution, (iii) at most one solution, over all* full *models of $E$?*

### 6.4 Equality of Solutions

In this section, we study the complexity of deciding whether terms have the same set of solutions over all (full) models. It is easy to see that the hardness of these problems is at least that of deciding equality in all (full) models. When considering all models, the problem turns out $\Pi^1_2$-complete, and, thus, higher than the degree $\Pi^1_1$ of equality in all models.

**Remark 2.** Let us briefly discuss the applicability of equality in all (full) models for the comparison of terms $s$, $t$ that are specified in independent specifications $E_s$ and $E_t$. First, we rename the symbols of one of the specifications such that $\Sigma_s \cap \Sigma_t = \{0,1,:\}$. Thereafter, we consider the validity of $s = t$ in the union $E_s \cup E_t$.

We show on two examples that this approach does not always yield the intended results. Let $E_M$ consist of the single equation $M = 1 : M$, and $E_N$ of

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

Then $M$ has the stream of ones as unique solution, but $N$ has no solution. Since $E_N$ does not have model, the union $E_M \cup E_N$ also does not admit one. Thus, $E_M \cup E_N \models M = N$ holds for trivial reasons. Nevertheless, we would not like to consider $M$ and $N$ as equivalent (at least if they are given by independent specifications).

Even if the specifications have unique solutions, a similar effect can occur. Let $M = \mathsf{zeros}$ and $E_M$ consist of the equations

$$\mathsf{is}_{\mathsf{zeros}}(\mathsf{nxor}(\sigma)) = \mathsf{zeros}$$
$$\mathsf{nxor}(0:0:\sigma) = 1 : \mathsf{nxor}(\sigma) \quad \mathsf{nxor}(0:1:\sigma) = 0 : \mathsf{nxor}(\sigma)$$
$$\mathsf{nxor}(1:0:\sigma) = 0 : \mathsf{nxor}(\sigma) \quad \mathsf{nxor}(1:1:\sigma) = 1 : \mathsf{nxor}(\sigma)$$

together with the equations (10). Let $N = \mathsf{blink}$ and $E_N$ consist of the equation $\mathsf{blink} = 0 : 1 : \mathsf{blink}$. Both specifications have models, and $\mathsf{zeros}$ and $\mathsf{blink}$ have unique solutions. For example, $E_M$ admits a model whose domain consists of all eventually constant streams. However, $E_M$ rules out models for which there exist elements $\sigma \in A_S$ with $[\![\mathsf{nxor}]\!](\sigma) = 0^\omega$. In particular, the stream $0101\ldots$ is excluded from the domain $A_S$. As a consequence, the union $E_M \cup E_N$ has no models, and $E_M \cup E_N \models \mathsf{zeros} = \mathsf{blink}$ holds.

As a consequence of the proof of Theorem 3, we obtain:

**Theorem 5.** *The following problem subsumes the analytical hierarchy:*

INPUT: *Bitstream specifications $E_s$, $E_t$, ground terms $s, t :: S$.*
QUESTION: *Do $s$ and $t$ have equal solutions over all* full *models, that is, $[\![s]\!]_{E_s,\,full} = [\![t]\!]_{E_t,\,full}$?*

We conclude this section with an investigation of the complexity of deciding whether two terms have the same set of solutions over all models. The proof of Theorem 1 yields only $\Pi^1_1$-hardness. In order to show $\Pi^1_2$-hardness, we employ a result of [4] stating that it is a $\Pi^1_2$-complete problem to decide whether the $\omega$-language of a non-deterministic Turing machine contains all words $\{0,1\}^\omega$.

Therefore, we consider non-deterministic Turing machines with one-sides tapes. Without loss of generality, we may restrict the non-determinism $\delta : Q \times \Gamma \to \wp(Q \times \Gamma \times \{L,R\})$ to binary choices in each step, that is, $|\delta(q,b)| \leq 2$ for every $q \in Q$ and $b \in \{0,1\}$. (Broader choices then are simulated by sequences of binary choices.) Moreover, for our purposes, it suffices to consider Turing machines that never halt. For the $\omega$-language, halting always corresponds to rejecting a run, and this rejection can be simulated by alternating moving forth and back eternally.

That is, a non-deterministic Turing machine $\mathsf{M} = \langle Q, q_0, \delta_0, \delta_1\rangle$ has two transition functions $\delta_0, \delta_1 : Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$ and we allow a non-deterministic choice between these functions in each step. Note that, for modeling non-determinism in an equational specifications, we cannot take the union of the specifications $E_{\langle Q,\,q_0,\,\delta_0\rangle}$ and $E_{\langle Q,\,q_0,\,\delta_1\rangle}$, since multiple equations having the same left-hand side do not model choice, but additional restrictions on the models of the specification. To this end, we introduce a third argument for the binary function symbols $q \in Q$ in Definition 5. This argument then governs the non-deterministic choice. In order to model one-sided tapes, we introduce a fourth argument that stores the position on the tape, and is increased, when moving right, and decreased, when moving left. That is, we adapt Definition 5 to:

$$q(x, b:y, i:z, p) = q'(b':x, y, z, 1:p)$$
$$q(a:x, b:y, i:z, 1:p) = q'(x, a:b':y, z, p)$$

for $\delta_i(q,b) = \langle q', b', R\rangle$ and $\delta_i(q,b) = \langle q', b', L\rangle$, respectively. We use $E^n_{\mathsf{M}}$ to denote this specification, and $R^n_{\mathsf{M}}$ for the corresponding term rewriting system. In the initial configuration, the third argument should be an underspecified stream, allowing for any non-deterministic choice. We pass $\mathsf{zeros}$ as fourth argument, thereby ensuring that the head cannot move to negative tape indices.

A *run* of $\mathsf{M}$ on an $\omega$-word $w \in \{0,1\}^\omega$ is a $R^n_{\mathsf{M}}$ rewrite sequence starting from a term $q_0(\mathsf{zeros}, \underline{w}, \underline{N}, \mathsf{zeros})$ where $N \in \{0,1\}^\omega$ determines the non-deterministic choices; here $\underline{w}$ is the term $w(0) : w(1) : \ldots$ A run of $\mathsf{M}$ is *complete* if every tape position $p \geq 0$ is visited (that is, positions right of the starting position), and it is *oscillating* if some tape position is visited infinitely often. A run is *accepting* if it is complete and not oscillating, that it, it visits every position $p \geq 0$ at least once, but only finitely often.

**Definition 19.** The $\omega$-language $\mathcal{L}^\omega(\mathsf{M})$ is the set of all $\omega$-words $w \in \{0,1\}^\omega$ such that $\mathsf{M}$ has an accepting run $w$.

We employ the following result, which follows from [4]:

**Theorem 6.** *The set* $\{\mathsf{M} \mid \mathcal{L}^\omega(\mathsf{M}) = \{0,1\}^\omega\}$ *is* $\Pi^1_2$*-complete.*

We are now ready for the proof of $\Pi^1_2$-completeness of equality of the set of solutions over all models. In the proof, we introduce a fifth argument for the symbol $q \in Q$ in $E^n_{\mathsf{M}}$ which enforces progress (productivity) and rules out exactly the oscillating runs.

**Theorem 7.** *The following problem is* $\Pi^1_2$*-complete:*
INPUT: *Bitstream specifications* $E_s$, $E_t$, *ground terms* $s, t :: S$.
QUESTION: *Do* $s$ *and* $t$ *have equal solutions over all models equal, that is,* $[\![s]\!]_{E_s} = [\![t]\!]_{E_t}$ *?*

*Proof.* Let $\mathsf{M} = \langle Q, q_0, \delta_0, \delta_1\rangle$ be a non-deterministic Turing machine. We reduce the problem in Theorem 6 to a decision problem for the equality of the set of solutions over all full models. We let $s = \mathsf{X}$ and define the specification $E_s$ to consist of:

$$q_0(\mathsf{zeros}, \mathsf{X}, \mathsf{N}, \mathsf{zeros}, P) = \mathsf{zeros} \tag{16}$$
$$\mathsf{natstr}(P) = \mathsf{ones} \tag{17}$$
$$q(x, b : y, i : z, p, 1 : v) = q'(b' : x, y, z, 1 : p, v) \quad \text{for } \delta_i(q,b) = \langle q', b', R\rangle \tag{18}$$
$$q(a : x, b : y, i : z, 1 : p, 1 : v) = q'(x, a : b' : y, z, p, v) \quad \text{for } \delta_i(q,b) = \langle q', b', L\rangle \tag{19}$$
$$q(x, y, z, 1 : p, 0 : v) = 0 : q(x, y, z, p, v) \tag{20}$$
$$q(x, y, z, 0 : p, 0 : v) = \mathsf{ones} \tag{21}$$
$$q(a : x, b : y, i : z, 0 : p, 1 : v) = \mathsf{ones} \quad \text{for } \delta_i(q,b) = \langle q', b', L\rangle \tag{22}$$

The equation (16) starts $\mathsf{M}$ on the stream $\mathsf{X}$ with non-deterministic choices governed by $\mathsf{N}$ and $P$ for enforcing progress. The streams $\mathsf{X}$ and $\mathsf{N}$ are unspecified, thus arbitrary. The equation (17) ensures that $[\![P]\!]$ contains infinitely many zeros. The equations (18) and (19) model the computation of $\mathsf{M}$ as discussed before, but now in each step removing the context $1 : \Box$ from the fifth argument. If the fifth argument starts with a 0, then (20) decrements the position counter (the fourth argument). Recall, the position counter determines how many steps the Turing machine $\mathsf{M}$ is permitted to move left. Thus, always eventually decrementing the counter rules out the oscillating runs. The equations (21) and (22) rule out models where the head move left of the envisaged progress $[\![P]\!]$.

It is important to note that for any non-oscillating run $\sigma$, we can define a function $p : \mathbb{N} \to \mathbb{N}$ such that after $p(n)$ steps, $\mathsf{M}$ visits only tape indices $\geq n$. Then an assignment $[\![P]\!] = 1^{p(0)}\,0\,1^{p(1)}\,0\,1^{p(2)}\,0\ldots$ in the model will permit this run to happen, that is, the head will never fall behind the envisaged progress and Equations (21) and (22) do not apply.

As a consequence, we have $[\![s]\!]_{E_s} = \{0,1\}^\omega$ if and only if for every $[\![\mathsf{X}]\!] \in \{0,1\}^\omega$ there exists a non-oscillating run (that is, an appropriate choice $[\![\mathsf{N}]\!]$) of $\mathsf{M}$ on $[\![\mathsf{X}]\!]$. Now we define $t = \mathsf{Y}$ and $E_t = \{\mathsf{Y} = \mathsf{Y}\}$ for which obviously $[\![t]\!]_{E_t} = \{0,1\}^\omega$. Therefore, $[\![s]\!]_{E_s} = [\![t]\!]_{E_t}$ if and only if $\mathcal{L}^\omega(\mathsf{M}) = \{0,1\}^\omega$. This concludes the proof of $\Pi^1_2$-hardness.

For $\Pi^0_2$-membership, the problem can be characterized by the following analytical formula: $\forall\langle\mathcal{A}_s, \mathcal{A}_t\rangle.\exists\langle\mathcal{A}'_s, \mathcal{A}'_t\rangle.(\mathcal{A}_s \models E_s \Rightarrow \mathcal{A}'_t \models E_t \wedge [\![s]\!]^{\mathcal{A}_s} = [\![t]\!]^{\mathcal{A}'_t}) \wedge (\mathcal{A}_t \models E_t \Rightarrow \mathcal{A}'_s \models E_s \wedge [\![t]\!]^{\mathcal{A}_t} = [\![s]\!]^{\mathcal{A}'_s})$. As in the proof of Theorem 1, here, it suffices to quantify over countable models. □

## 7. Equality for Behavioral Specifications

In this section we consider the notion of equality from [17] which is based on hidden algebras [16]. We introduce the hidden models of bitstream specifications as employed in [17], where it has been shown that deciding the equality of (equationally defined) streams, with respect to this semantics, is a $\Pi^0_2$-complete problem. We consider the following two extensions of this semantics:

(i) extending the semantics to streams over natural numbers, or

(ii) requiring the behavioral equivalence $\equiv$ to be a congruence.

We show that both extensions lift the complexity of deciding equality to the level $\Pi^1_1$ of the analytical hierarchy. If the specifications are required to be productive (thus, separating the problem of productivity [10] from that of equality) it can be shown that the complexity resides at $\Pi^0_1$ [13]. The results in [17] (as well as the results we mention in the current paper) are based on the comparison of non-productive specifications, and the proofs inherently encode productivity problems.

Let us briefly explain why the $\Pi^1_1$-completeness for the equality of bitstreams in Theorem 1 does not directly carry over the setup of [17]. The problem is the definition of the function $\mathsf{natstr}$ in (12) containing the equation $\mathsf{natstr}(\mathsf{ones}) = \mathsf{zeros}$. This equation does not work if we have confusion in the models and behavioral equivalence is not a congruence. In particular, as discussed in Section 2, if $\mathsf{ones}' = 1 : \mathsf{ones}'$, we cannot conclude that $\mathsf{natstr}(\mathsf{ones}') = \mathsf{zeros}$. As a consequence, with the behavioral specifications of [17] it is not possible to enforce that a bitstream always eventually contains a zero. However, if we consider behavioral specifications of streams of natural numbers, then we no longer need $\mathsf{natstr}$, hence, reestablishing the $\Pi^1_1$-completeness result for the equality of streams of natural numbers specified behaviorally. There is a similar problem with the equation $\mathsf{is}_{\mathsf{zeros}}(\mathsf{zeros}) = \mathsf{ones}$, that, however, can be overcome by discarding $\mathsf{is}_{\mathsf{zeros}}$ as in the proof of Theorem 2.

### 7.1 Basic Setup

In [17], every bitstream specification contains the equations

$$\mathsf{head}(x : \sigma) = x \qquad\qquad \mathsf{tail}(x : \sigma) = \sigma$$

where $\mathsf{head} :: S \to B$ and $\mathsf{tail} :: S \to S$.

**Definition 20.** A *hidden* $\Sigma$*-algebra* $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ consists of

(i) an $\mathcal{S}$-sorted domain $A$ where $A_B = \{0,1\}$,
(ii) for every $f :: s_1 \times \ldots \times s_n \to s \in \Sigma$ an *interpretation* $[\![f]\!] : A_{s_1} \times \ldots A_{s_n} \to A_s$,
(iii) $0, 1 \in \Sigma$ with $[\![0]\!] = 0$ and $[\![1]\!] = 1$.

We stress that now $A_S$ is an *arbitrary* set.

**Definition 21.** Let $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ be a hidden $\Sigma$-algebra. Then $\sigma, \tau \in A_S$ are called *behaviorally equivalent*, denoted by $\sigma \equiv \tau$, if they are indistinguishable with $\{\mathsf{head}, \mathsf{tail}\}$-experiments, that is:

$$\sigma \equiv \tau \iff \forall n \in \mathbb{N}.\ [\![\mathsf{head}]\!]([\![\mathsf{tail}]\!]^n(\sigma)) = [\![\mathsf{head}]\!]([\![\mathsf{tail}]\!]^n(\tau)$$

On the domain $A_B$, we let $\equiv$ be the identity relation.

Note that $\equiv$ is a not a congruence (only for $[\![\mathsf{head}]\!]$ and $[\![\mathsf{tail}]\!]$).

**Definition 22.** Let $E$ be a bitstream specification over $\Sigma$. A hidden $\Sigma$-algebra *behaviorally satisfies* $E$, denoted $\mathcal{A} \models\!\!\models E$, if for every equation of $E$, the left- and right-hand sides are behaviorally equivalent: $[\![\ell]\!]_\alpha \equiv [\![r]\!]_\alpha$ for every $\ell = r \in E$ and $\alpha : \mathcal{X} \to A$. We say that an equation $\ell = r$ is *behaviorally satisfied in all hidden models of* $E$, denoted $E \models\!\!\models \ell = r$ if $\mathcal{A} \models\!\!\models E$ implies $\mathcal{A} \models\!\!\models \ell = r$ for every hidden $\Sigma$-algebra $\mathcal{A}$.

For a discussion of this semantics, we refer to Section 2.

### 7.2 Behavioral Equivalence as Congruence

We now adapt the basic setup by requiring $\equiv$ to be a *congruence relation*, that is, $s \equiv t$ implies $f(\ldots, s, \ldots) \equiv f(\ldots, t, \ldots)$. The resulting models are called *behavioral* in [2].

**Definition 23.** A hidden $\Sigma$-algebra is called *behavioral* if $\equiv$ is a congruence relation. For a bitstream specification $E$ over $\Sigma$, we say that $\ell = r$ *is behaviorally satisfied in all behavioral models of* $E$ if $\mathcal{A} \models E \Rightarrow \mathcal{A} \models \ell = r$ for every behavioral hidden $\Sigma$-algebra $\mathcal{A}$.

**Theorem 8.** *The following problem is $\Pi^1_1$-complete:*
INPUT: *Bitstream specification $E$, terms $s, t :: S$.*
QUESTION: *Is $s = t$ satisfied in all behavioral models of $E$?*

*Proof.* We show: the equation $s = t$ is behaviorally satisfied in all behavioral models of $E$ if and only if $s = t$ holds in all models of $E$; the latter property is $\Pi^1_1$-complete by Theorem 1.

The direction '$\Leftarrow$' follows immediately, since every $\Sigma$-algebra is a behavioral hidden $\Sigma$-algebra. For '$\Rightarrow$', let $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ be a hidden $\Sigma$-algebra. Let $\mathcal{A}/_{\equiv} = \langle A/_{\equiv}, [\![\cdot]\!]/_{\equiv}\rangle$ be the quotient algebra. That is, $A/_{\equiv}$ are the congruence classes of $A$ with respect to $\equiv$. For symbols $f \in \Sigma$ and $B_1, \ldots, B_{ar(f)} \in A/_{\equiv}$, we define $[\![f]\!]/_{\equiv}(B_1, \ldots, B_{ar(f)}) = B$ if $[\![f]\!](b_1, \ldots, b_{ar(f)}) = b$ for $b_1 \in B_1, \ldots, b_{ar(f)} \in B_{ar(f)}$, and $B$ is the congruence class of $b$ with respect to $\equiv$. The quotient algebra $\mathcal{A}/_{\equiv}$ is a behavioral hidden $\Sigma$-algebra that, due to $\equiv$ being a congruence, behaviorally satisfies the same equations as $\mathcal{A}$. Let $\mathcal{A}'$ be the $\Sigma$-algebra obtained from $\mathcal{A}/_{\equiv}$ by renaming the domain elements into the streams they represent, that is, $a \in (\mathcal{A}/_{\equiv})_S$ becomes $[\![\mathsf{head}]\!](a) : [\![\mathsf{head}]\!]([\![\mathsf{tail}]\!](a)) : \ldots$. Then $[\![:]\!](x, \sigma) = x : \sigma$, since in $\mathcal{A}/_{\equiv}$ every stream has a unique representative in the model. Hence, $\mathcal{A}'$ is a stream algebra. Moreover, for elements $a, b$ of the domain of $\mathcal{A}/_{\equiv}$, we have $a \equiv b$ iff $a = b$. Hence, $\mathcal{A}'$ is a model of an equation $s = t$ if and only if $s = t$ is behaviorally satisfied in $\mathcal{A}$. □

### 7.3 Streams of Natural Numbers

We briefly study hidden models with confusion, described in Section 2, for streams of natural numbers. A $\mathbb{N}$-*stream specification* is now defined like a bitstream specification, except the sorts are $\mathcal{S} = \{N, S\}$, and the symbols are $0 :: N$, $s :: N \to N$ and ':' of type $N \times S \to S$. We adapt the definition of hidden $\Sigma$-algebras accordingly.

**Definition 24.** A *hidden $\Sigma$-algebra* $\mathcal{A} = \langle A, [\![\cdot]\!]\rangle$ consists of

(i) an $\mathcal{S}$-sorted domain $A$ and $A_N = \mathbb{N}$,
(ii) for every $f :: s_1 \times \ldots \times s_n \to s \in \Sigma$ an *interpretation* $[\![f]\!] : A_{s_1} \times \ldots A_{s_n} \to A_s$,
(iii) $0, s \in \Sigma$ with $[\![0]\!] = 0$ and $[\![s]\!](x) = x + 1$,
(iv) for every $s \in A_S$ there are $n \in \mathbb{N}$ and $s' \in A_S$ such that we have $s = [\![:]\!](b, s')$; see further Remark **??**.

The definitions of behavioral equivalence and satisfaction are the same as for bitstream specifications. A slight modification of the proof of Theorem 2 results in the following.

**Theorem 9.** *The following problem is $\Pi^1_1$-complete:*
INPUT: *$\mathbb{N}$-stream specification $E$, terms $s, t :: S$.*
QUESTION: *Does $E \models s = t$ hold? That is, is $s = t$ behaviorally satisfied in all hidden models of $E$?*

*Proof.* We reduce the well-foundedness problem for decidable binary relations to an equality problem. Let $M \subseteq \mathbb{N} \times \mathbb{N}$ be a decidable predicate, and $\mathsf{M} = \langle Q, q_0, \delta\rangle$ the corresponding Turing machine. We define the following specification $E$:

$$\mathsf{zeros} = \mathsf{run}(1, \mathsf{X}) \qquad \mathsf{unary}(0) = \mathsf{zeros}$$
$$\mathsf{run}(0, \sigma) = \mathsf{ones} \qquad \mathsf{unary}(s(x)) = 1 : \mathsf{unary}(x)$$
$$\mathsf{run}(1, \sigma) = 0 : \mathsf{run}(\mathsf{M}(\mathsf{zeros}; \mathsf{unary}(\mathsf{head}(\sigma)), \mathsf{unary}(\mathsf{head}(\mathsf{tail}(\sigma)))), \mathsf{tail}(\sigma))$$

together with the equations from $E_{\mathsf{M}}$ and (9). In contrast with the proof of Theorem 2, $\mathsf{X}$ is now a stream of natural numbers. Since $\mathsf{X}$ is unspecified, its interpretation in the model can be an arbitrary stream of natural numbers. As in the proofs of Theorems 1 and 2, we employ $\mathsf{X}$ to guess an infinite path through $M$. Instead of $\mathsf{uhd}(\cdot)$ and $\mathsf{utl}(\cdot)$ on bitstreams, we now take $\mathsf{unary}(\mathsf{head}(\cdot))$ and $\mathsf{tail}(\cdot)$, respectively, where the function $\mathsf{unary}$ converts natural numbers to unary representations in forms of streams. As in the proof of Theorem 2, it follows that there exists a hidden $\Sigma$-algebra $\mathcal{A}$ with $\mathcal{A} \models E$ if and only if $M$ is not well-founded. Thus, $E \models \mathsf{zeros} = \mathsf{ones}$ if and only if $M$ is well-founded. □

## 8. Equivalence of Lambda Terms

In this section we investigate the complexity of deciding the equality of $\lambda$-terms with respect to the observational equivalences $=_{nf}$, $=_{hnf}$ and $=_{whnf}$ as introduced in Section 1. Furthermore, we study the complexity of deciding whether two $\lambda$-terms have the same Böhm trees or Lévy–Longo trees. The interested reader is referred to [1, 7] for an introduction to Böhm trees, and to [6] for a thorough study of the observational equivalences on $\lambda$-terms.

**Definition 25.** Let $M$ be a $\lambda$-term. The *Böhm tree* $\mathsf{BT}(M)$ *of* $M$ is a potentially infinite term defined as follows. If $M$ has no hnf, then $\mathsf{BT}(M) = \bot$. Otherwise, there is a head reduction $M \to_h^* \lambda x_1. \ldots \lambda x_n. y M_1 \ldots M_m$ to head normal form. Then we define $\mathsf{BT}(M) = \lambda x_1. \ldots \lambda x_n. y \mathsf{BT}(M_1) \ldots \mathsf{BT}(M_m)$.

**Definition 26.** Let $M$ be a $\lambda$-term. The *Lévy–Longo tree* $\mathsf{LT}(M)$ *of* $M$ is a potentially infinite term defined as follows:

$$\mathsf{LT}(M) = \bot \qquad \text{if } M \text{ has no whnf}$$
$$\mathsf{LT}(M) = \lambda x.\mathsf{LT}(N) \qquad \text{if } M \to_h^* \lambda x.N$$
$$\mathsf{LT}(M) = x\mathsf{LT}(M_1) \ldots \mathsf{LT}(M_m) \qquad \text{if } M \to_h^* x M_1 \ldots M_m$$

For the observational equivalences we obtain:

**Theorem 10.** *For each $=_? \in \{=_n, =_h, =_w\}$, the following problem is $\Pi^0_2$-complete:*
INPUT: *$\lambda$-terms $M$, $N$.*
QUESTION: *Does $M =_? N$ hold?*

*Proof.* First, we show $\Pi^0_2$-membership of the problem. We consider $=_n$ ($=_h$ and $=_w$ work analogously). A $\lambda$-term $Q$ has a normal form if and only if $Q$ admits a standard reduction $\to_{std}^*$ to a normal form, see [1]. For a $\lambda$-term $Q$, and $n \in \mathbb{N}$, we write $Q \to_{std}^{\leq n} nf$ to denote that $Q$ rewrites to a normal form within $\leq n$ steps of standard reduction. Note that this is a decidable property. Then we claim:

$$M =_n N \iff \tag{23}$$
$$\forall C.\, \forall n.\, \exists m.\, \left( C[M] \to_{std}^{\leq n+m} nf \Leftrightarrow C[N] \to_{std}^{\leq n+m} nf \right)$$

For '$\Rightarrow$' in (23), assume that $M =_n N$. Let $C$ be a context. We distinguish the following cases:

(i) Assume that $C[M]$ has a normal form. Then $C[N]$ has one, and $C[M] \to_{std}^{k} nf$ and $C[N] \to_{std}^{\ell} nf$ for some $k, \ell \in \mathbb{N}$. Then in (23) for any $n \in \mathbb{N}$ we can choose $m = \max(k, \ell)$.
(ii) The case that $C[N]$ has a normal form is symmetric to (i).
(iii) If neither $C[M]$ nor $C[N]$ have a normal form, then neither $C[M] \to_{std}^{\leq n+m} nf$ nor $C[N] \to_{std}^{\leq n+m} nf$ for any $n, m \in \mathbb{N}$.

For '$\Leftarrow$' in (23), assume $M \neq_n N$. Then there is a context $C$ such that exactly one of the terms $C[M]$ and $C[N]$ has a normal form; without loss of generality, assume $C[M] \to_{std}^{\leq n} nf$ for some $n \in \mathbb{N}$. Hence, $C[M] \to_{std}^{\leq n+m} nf$ for every $m \in \mathbb{N}$, but $C[N] \to_{std}^{\leq n+m} nf$ for no $m \in \mathbb{N}$. Thus, the right-hand side of (23) is not satisfied.

From (23) it follows that $=_n$ is in $\Pi^0_2$, since the two quantifiers $\forall C$ and $\forall n$ can be merged into a single $\forall$-quantifier.

We now proceed with proving $\Pi^0_2$-hardness of the problem. Let $\mathsf{T}$ be a Turing machine, and let $T$ be a $\lambda$-term such that for all $n, m \in \mathbb{N}$, $T\,\underline{n}\,\underline{m}$ rewrites to $\mathsf{K}$ if $\mathsf{T}$ terminates on input $n$ within $m$ steps, and to $\mathsf{KI}$, otherwise. Here, $\mathsf{K} = \lambda xy.x$ and $\mathsf{I} = \lambda x.x$ are the usual combinators, and $\underline{k} = \lambda f.\lambda x.f^n x$ is the Church numeral representing the natural number $k \in \mathbb{N}$. The construction of such $T$ is standard, see [1]. Now we define:

$$M = (\lambda x.\lambda a.a(xx))(\lambda x.\lambda a.a(xx))$$
$$N = N'N'\mathsf{zer} \qquad N' = \lambda xn.T'n\,\mathsf{zer}(\lambda a.a(xx(\mathsf{succ}\,n)))$$
$$T' = T''T'' \qquad T'' = \lambda xnm.Tnm\mathsf{I}(xxn(\mathsf{succ}\,m))$$
$$\mathsf{zer} = \lambda fx.x \qquad \mathsf{succ} = \lambda zfx.f(zfx)$$

We show that $M =_? N$ if and only if $\mathsf{T}$ halts on all $n \in \mathbb{N}$. Note that $T'\underline{n}\,\underline{m} \to^* \mathsf{I}$ if $T\underline{n}\,\underline{m} \to^* \mathsf{K}$, that is, if $\mathsf{T}$ terminates on input $n$ in $m$ steps; otherwise $T'\underline{n}\,\underline{m} \to^* T'\underline{n}\,(\underline{m+1})$. Hence, we obtain

$$T'\underline{n}\,\underline{0} \to^* \mathsf{I} \iff \mathsf{T} \text{ halts on input } n$$
$$\iff T'\underline{n}\,\underline{0} \text{ has a (weak) head normal form}$$

The Lévy–Longo tree of $M$ is $\lambda a.a(\lambda a.a(\lambda a.a\ldots))$. If $\mathsf{T}$ halts on input $n$, we have

$$N'N'\underline{n} \to^* T'\underline{n}\,\mathsf{zer}(\lambda a.a(N'N'\underline{n+1})) \to^* \lambda a.a(N'N'\underline{n+1})$$

Thus if $\mathsf{T}$ terminates on all $n \in \mathbb{N}$, then the Lévy–Longo trees of $M$ and $N$ are equal, and, hence, by [6] we have $M =_w N$, $M =_h N$ and $M =_n N$. Otherwise, let $n \in \mathbb{N}$ be minimal such that $\mathsf{T}$ does not halt on $n$. Then by the above, we have:

$$N \to^* \underbrace{\lambda a.\,a(\lambda a.\,a(\ldots \lambda a.\,a}_{n\text{-times}}(N'N'\underline{n})\ldots))$$

Then $N\mathsf{I}^n \to^* N'N'\underline{n}$ has no (weak) head normal form, but $M\mathsf{I}^n$ has. Thus we have $M \neq_n N$, $M \neq_h N$ and $M \neq_w N$. This proves $\Pi^0_2$-hardness. □

The proof immediately yields the following result:

**Theorem 11.** *The following problems are $\Pi^0_2$-complete:*
INPUT: *$\lambda$-terms $M$, $N$.*
QUESTION: *(i) Do $s$ and $t$ have equal Böhm trees?*
*(ii) Do $s$ and $t$ have equal Lévy–Longo trees?*

*Proof.* Follows immediately from the proof of Theorem 10 since $M$ and $N$ are observationally equal if and only if they have the same Lévy–Longo tree, and for $M$ and $N$ the Lévy–Longo trees coincide with their Böhm trees. □

We mention that for Berarducci trees, the proof of Theorem 10 implies $\Pi^0_2$-hardness. It is not difficult to see that the problem of deciding the equality of Berarducci trees is in $\Pi^0_3$. We leave the determination of the precise complexity to future work.

## 9. Conclusions

We have investigated different model-theoretic and rewriting based semantics of equality of infinite objects, specified either by systems of equations or by $\lambda$-terms. It turns out that the complexities for these notions vary from the low levels of the arithmetical hierarchy $\Pi^0_1$ and $\Pi^0_2$, up to $\Pi^1_1$ and $\Pi^1_2$ of the analytical hierarchy, and some even subsume the entire arithmetical and analytical hierarchy. In particular, the observational equivalences of $\lambda$-terms, that are of interest for functional programming, are all $\Pi^0_2$-complete.

Apart from $\Pi^0_1$, none of these classes are recursively enumerable or co-recursively enumerable. Thus, there exists no complete proof systems for proving or for disproving equality. An exception is the equality of normal forms for productive specifications for which inequalities can be recursively enumerated [13].

## References


[1] H. P. Barendregt. *The Lambda Calculus, its Syntax and Semantics*. North-Holland, 1984.

[2] M. Bidoit, R. Hennicker, and A. Kurz. Observational Logic, Constructor-based Logic, and Their Duality. *Theor. Comput. Sci.*, 298:471–510, 2003.

[3] S. R. Buss and G. Rosu. Incompleteness of Behavioral Logics. *ENTCS*, 33:61–79, 2000.

[4] J. Castro and F. Cucker. Nondeterministic $\omega$-Computations and the Analytical Hierarchy. *Logik u. Grundlagen d. Math*, 35:333–342, 1989.

[5] T. Coquand. Infinite Objects in Type Theory. In *Postproc. Conf. on Types for Proofs and Programs (TYPES 1993)*, volume 806 of *LNCS*, pages 62–78. Springer, 1993.

[6] M. Dezani-Ciancaglini and E. Giovannetti. From Böhm's Theorem to Observational Equivalences: an Informal Account. In *BOTH'01*, volume 50 of *ENTCS*, 2001.

[7] M. Dezani-Ciancaglini, P. Severi, and F.-J. de Vries. Böhm's theorem for Berarducci trees. In *CATS 2000 Computing: the Australasian Theory Symposium*, volume 31 of *ENTCS*, 2000.

[8] J. Endrullis, H. Geuvers, J. G. Simonsen, and H. Zantema. Levels of Undecidability in Rewriting. *Information and Computation*, 209(2):227–245, 2011.

[9] J. Endrullis, C. Grabmayer, and D. Hendriks. Data-Oblivious Stream Productivity. In *Proc. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2008)*, number 5330 in LNCS, pages 79–96. Springer, 2008.

[10] J. Endrullis, C. Grabmayer, and D. Hendriks. Complexity of Fractran and Productivity. In *Proc. Conf. on Automated Deduction (CADE 22)*, volume 5663 of *LNCS*, pages 371–387, 2009.

[11] D. P. Friedman and D. S. Wise. CONS Should Not Evaluate its Arguments. In *ICALP*, pages 257–284, 1976.

[12] H. Geuvers. Inductive and Coinductive Types with Iteration and Recursion. In *Proc. Workshop on Types for Proofs and Programs (TYPES 1992)*, pages 193–217, 1992.

[13] C. Grabmayer, J. Endrullis, D. Hendriks, J. W. Klop, and L. S. Moss. Automatic Sequences and Zip-Specifications. In *Proc. Symp. on Logic in Computer Science (LICS 2012)*. IEEE Computer Society, 2012. To appear.

[14] P. Henderson and J. H. Morris, Jr. A Lazy Evaluator. In *Proc. ACM SIGACT-SIGPLAN Symp. on Principles on programming languages (POPL)*, pages 95–103. ACM, 1976.

[15] G. Malcolm. Hidden Algebra and Systems of Abstract Machines. In *Proc. Symp. on New Models for Software Architecture (IMSA)*, 1997.

[16] G. Roşu. *Hidden Logic*. PhD thesis, University of California, 2000.

[17] G. Roşu. Equality of Streams is a $\Pi^0_2$-complete Problem. In *Proc. ACM SIGPLAN Conf. on Functional Programming (ICFP)*, pages 184–191. ACM, 2006.

[18] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York, 1967.

[19] J. J. M. M. Rutten. Behavioural Differential Equations: a Coinductive Calculus of Streams, Automata, and Power Series. *Theor. Comput. Sci.*, 308(1-3):1–53, 2003.

[20] J. J. M. M. Rutten. A Tutorial on Coinductive Stream Calculus and Signal Flow Graphs. *Theor. Comput. Sci.*, 343:443–481, 2005.

[21] D. Sangiorgi and J. J. M. M. Rutten. *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2012.

[22] J. R. Shoenfield. *Degrees of Unsolvability*. North-Holland, 1971.

[23] B. A. Sijtsma. On the Productivity of Recursive List Definitions. *ACM Transactions on Programming Languages and Systems*, 11(4):633–649, 1989.

[24] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

[25] M. Walicki and S. Meldal. Nondeterminism vs. underspecification. In *Proc. of the World Multiconference on Systemics, Cybernetics and Informatics*, ISAS-SCI 2001, pages 551–555. IIIS, 2001.