Well-definedness and observational equivalence for inductive-coinductive programs

Basold, Henning; Hansen, Helle Hvid

DOI
10.1093/logcom/exv091

Publication date
2019

Document Version
Accepted author manuscript

Published in
Journal of Logic and Computation

Citation (APA)
Basold, H., & Hansen, H. H. (2019). Well-definedness and observational equivalence for inductive-coinductive programs. *Journal of Logic and Computation*, *29*(4), 419-468. https://doi.org/10.1093/logcom/exv091

Well-definedness and Observational Equivalence for Inductive-Coinductive Programs

Henning Basold and Helle Hvid Hansen

Abstract

In this paper, we investigate well-definedness and observational equivalence for programs of mixed inductive and coinductive types. The programs we consider are terms of a variation on the copattern language introduced by Abel et al. These notions are defined by means of tests formulas which combine structural congruence for inductive types and modal logic for coinductive types. Tests also correspond to certain evaluation contexts. We define a program to be well-defined if it is strongly normalising under all tests, and two programs are observationally equivalent if they satisfy the same tests.

We show that observational equivalence is strong enough to ensure that least and greatest fixed point types are initial algebras and final coalgebras, respectively. This yields inductive and coinductive proof principles for reasoning about program behaviour. On the other hand, we argue that observational equivalence does not identify too many terms, by showing that tests induce a topology that coincides with the widely accepted topology used to define computable functions on coinductive types as continuous functions.

As one would expect, observational equivalence is, in general, undecidable, but in order to develop some practically useful heuristics we provide coinductive techniques for establishing observational normalisation and observational equivalence, along with up-to techniques for enhancing these methods.

"Everything that is observed in the whole of nature must be able to be deduced from it."

— B. Spinoza, Principles of Cartesian Philosophy

1 Introduction

There is a growing interest in programming with mixed inductive-coinductive types [2, 3, 14, 16, 17, 20, 21, 33, 34], especially in combination with specifications of programs by means of (co)recursive equations in the style of behavioural differential equations [43]. We address two fundamental questions about such programs: First, when is a program well-defined, i.e., when does it have a well-defined semantics? Second, when are two programs considered behaviourally equivalent? Both questions are particularly important in type-based theorem provers, as well-definedness ensures logical consistency [31], and a strong program equivalence eases reasoning [6].

For programs specified by systems of (co)recursive equations, well-definedness amounts to existence and uniqueness of solutions. For purely corecursive equations well-definedness is often referred to as productivity, and it is usually ensured by requiring that the equations are of a certain syntactic shape. Examples include the (abstract) GSOS format of [46] and the guardedness condition of [5]. Such syntactic conditions are easy to check, and are therefore used in theorem provers like Coq, cf. [17]. The problem, as argued in e.g. [1], is that they are very restrictive and not easy to extend in a compositional manner.

Equivalences of terms in (lazy) functional languages have been considered in e.g. [4, 6, 19, 23, 24, 32, 35, 37]. We discuss their relation to the present work in more detail below. The main difference to mention now is that coinductive types are not considered there. The only work we are aware of on equivalences in type systems with mixed inductive and coinductive types is by Plotkin & Abadi [36] and Hasegawa [22] which uses encoding of fixed point types in the polymorphic λ-calculus. We contribute to this line of research by investigating program equivalence in type systems in which inductive and coinductive types are first-class citizens. To the best of our knowledge, this has not been done before.

In this paper, we define notions of productivity and behavioural equivalence in a simply typed, lazy language with least and greatest fixed point types. This language is a variation of the one in [3] where copatterns are introduced to program on infinite data. A copattern equation is essentially a behavioural differential equations which also allows pattern matching.

Our perspective is coalgebraic [42], that is, based on the notion of observable behaviour and bisimilarity. Informally, two programs are behaviourally equivalent if they cannot be distinguished by any observation, and a term is productive if every observation terminates with a well-defined result. We formalise this by defining *test formulae* that capture sequences of observations on programs. These tests correspond to certain evaluation contexts, and are defined in a manner similar to the testing logic of [44, Sec. 6.2] and the coalgebraic modal logics for Kripke-polynomial functors in [40, 26]. For terms of inductive type, tests inspect terms by matching on constructors and testing subterms. On coinductive types, tests are essentially modal formulas. We call a program *observationally normalising (or productive)* if it is strongly normalising under all tests, and two programs are *observationally equivalent* if they satisfy the same tests. For terms of function type this amounts to requiring that on observationally normalising input, the outputs are observationally equivalent.

We argue for the suitability of observational equivalence by showing that identifying observationally equivalent functions yields coinduction and induction principles for reasoning about programs. More precisely, we construct a category $\mathbb{T}^{\downarrow}$ that has types as objects and terms of function type modulo observational equivalence as arrows. On this category we define functors from types such that least and greatest fixed point types are initial algebras and final coalgebras, respectively (Thm. 4.11). Thus observationally equivalence is sufficiently strong to capture the relevant coalgebraic and algebraic notions of equivalence. On the other hand, we argue that observational equivalence does not identify too many terms by showing that tests induce a topology (Prop. 4.16) that coincides with the widely accepted topology used to define computable functions on coinductive types as continuous functions. This, together with the categorical properties, provides evidence that observational equivalence is a good semantical notion.

As one would expect, observational equivalence is, in general, undecidable which we prove by reduction to the Post correspondence problem. An immediate question is whether it becomes decidable over a restricted part of the language, which we answer positively by showing that observational equivalence is decidable over the "data fragment" (i.e., function-free fragment). In order to develop some practically useful heuristics for the full language, we provide coinductive techniques for establishing observational normalisation and observational equivalence. These techniques are obtained by defining a transition system of terms on which observational equivalence coincides with bisimilarity, and observational normalisation is a coinductive predicate. An advantage of these coalgebraic characterisations is, moreover, that they facilitate the use of up-to techniques that lead to a significant simplification of proofs and are compositional. Specifically, we provide up-to convertibility and up-to evaluation techniques, and we demonstrate with an example how up-to techniques can be derived for program definitions in a format that generalises the simple SOS format [29].

Related Work. The idea of categorical data types and unique mapping properties has previously been proposed by, e.g., Hagino [20] and Jacobs [25, Sec. 2.3]. These approaches are centred around postulating the equations that ensure the unique mapping properties. The hard part is to decide term equality induced by these equations. This can be achieved, e.g., by casting the equations into a strongly normalising and confluent rewriting system cf. Ghani [15] and Curien & Cosmo [11]. In contrast, we get unique mapping properties by taking observational equivalence as equality, and we obtain (co)inducive proof principles for reasoning about term equivalence.

Type systems with inductive and coinductive types have been studied via encoding into the (impredicative) polymorphic λ-calculus, see e.g. Geuvers [14]. Using this approach, Plotkin & Abadi [36] and Hasegawa [22] have shown that parametricity schemes induce an equivalence which yields initial algebras and final coalgebras. We avoid such an encoding into an impredicative language, and reason in the “native” language.

Program equivalence in a dependent type theory is formalised in Altenkirch et al. [6] via a propositional equality type called *observational equality.* This equality type is defined inductively over types in a manner similar to ours, but they do not need to require (an analogue of) observational normalisation as their term language contains only restricted forms of λ-abstraction and recursion which ensures that all terms are strongly normalising. In particular, their system has no coinductive types. We, on the other hand, have a more flexible term language which allows definitions via systems of copattern equations.

Our notion of observational equivalence is similar to the *contextual equivalence* of Plotkin [37] and Milner [32], since test formulae can be interpreted as certain program contexts. Similar in spirit is also Zeilbergers's observational equivalence [47], as he considers programs to be equivalent if they yield the same result in all environments. Interestingly, he starts by considering only the result that programs diverge, and then observes that, in order to get a useful equivalence, he needs to have a second possible result. This is similar to the fact that our test language has two basic tests, namely the everywhere true and the everywhere false test. Without these, we would get a trivial equivalence as well.

The *observational congruence* of Pitts [35] characterises contextual equivalence for an extension of PCF as the greatest relation with certain properties. One of these properties (called *compatibility*) is an inductive predicate on relations, hence observational congruence is different from our coinductive notion of equivalence. We can show that observational equivalence is contained in Pitts' observational congruence when suitably adopted to our setting.

For inductive types and abstractions, observational equivalence is also closely related to the notion of applicative bisimilarity in the lazy λ-calculus by Abramsky [4], which compares terms by reducing to a weak head normal form (with a λ in head position) and comparing the results of function application. It is possible to extend applicative bisimilarity in a natural way to all our terms, but the resulting notion will differ from observational equivalence. In particular, applicative bisimilarity is weaker on diverging terms, as also discussed in [19].

Our work is closest in spirit to the work by Howe [23, 24] and Gordon [19]. Howe [23, 24] studies coinductive proof princples for reasoning about equivalence in a general class of lazy languages with binding, and discusses applications in intuitistic type theory and the Nuprl theorem prover. Unique mapping properties are not considered. Gordon [19] gives a bisimulation characterisation of contextual equivalence in the language FPC, and studies up-to techniques. Again, coinductive types are not part of FPC, and unique mapping properties are not investigated.

Outline. The rest of the paper is structured as follows. In Section 2, we introduce types, terms and a reduction relation. We show moreover that this reduction relation is confluent, which allows us to define convertibility of terms. In Section 3, we define tests, observational normalisation and observational equivalence, and prove some basic properties. In Section 4, we define the term category $\mathbb{T}^{\downarrow}$, and show that least and greatest fixed point types are initial algebras and final coalgebras, respectively, for functors defined from types, thus obtaining induction and coinduction principles for programs. We also study the properties of the topology associated with observational equivalence. In Section 5, we define the coalgebra of terms on which observational equivalence is bisimilarity and observational normalisation is a coinductive predicate, and we establish various up-to techniques that enhance these coinductive proof principles to become practical and useful. In Section 6, we prove the undecidability of observational equivalence, and decidability over the function-free fragment. We finish with concluding remarks and discuss directions for future research in Section 7.

2 Types and Terms

In this section we introduce our type system, the terms inhabiting these types, and a reduction relation on terms. In Abel et al. [3], a functional language was defined in which copatterns are used to specify coinductive data. Our language is based hereon with one modification: We split fixed point types from sum and product types, which eases the categorical reasoning.

2.1 Type Syntax

We assume given a countable set of type variables $X, Y, \ldots$. Types are defined relative to a type context Ξ which is a finite sequence of type variables that contains all free variables. More precisely, the type syntax is defined by the following rules together with the usual weakening, contraction and exchange rules.

$$\frac{}{\Xi, X \Vdash X} \qquad \frac{\Xi \Vdash A \qquad \Xi \Vdash B}{\Xi \Vdash A + B} \qquad \frac{\Xi \Vdash A \qquad \Xi \Vdash B}{\Xi \Vdash A \times B}$$

$$\frac{\Vdash A \qquad \Xi \Vdash B}{\Xi \Vdash A \to B} \qquad \frac{\Xi, X \Vdash A}{\Xi \Vdash \mu X.A} \qquad \frac{\Xi, X \Vdash A}{\Xi \Vdash \nu X.A}$$

A type is said to be *closed* if it is defined in an empty context, and we will simply write A instead of $\Vdash A$. The set of all closed types is denoted by Ty, and unless stated otherwise, types are assumed to be closed, which will be the case in the most part of the paper. The exception is in Sec. 4 where we need to make the context explicit.

We refer to sum and least fixed point types as *inductive types*, and to product, function and greatest fixed point types as *coinductive types*. Typical examples are the one type $1 := \nu X.X$, natural numbers $\mathrm{Nat} := \mu X.1 + X$; streams over A given by $\mathrm{Str}\, A := \nu X.A \times X$; and "left fair streams" $\mathrm{LFair}\, A\, B := \nu X.\mu Y.(A \times X + B \times Y)$, which are infinite sequences over A interleaved by finite lists over B.

Note that we can only construct *strictly positive* types by these rules, that is, types where type variables never occur left of an $\to$. There are two reasons for restricting ourselves to strictly positive types. The first is that we can only guarantee the correctness of Def. 3.1 for strictly positive types, not for positive or let alone arbitrary types. The second is that we otherwise could form the type $\mu X.X \to X$, allowing us to type any λ-term (see [7, Sec. 9.3]). This would cause fixed points to coincide, preventing an interpretation over categories like **Set** or, more generally, over algebraically non-compact categories.

We consider types to be equal up to α-equivalence, i.e., renaming of bound variables. See [7, Sec. 7] for a discussion of stronger notions of type equality.

2.2 Terms and Type Judgment

Next, we define the syntax and typing of terms, patterns and copatterns. Formal typing judgments are provided in Def. 2.1 below, but we first give an informal description and introduce terminology.

The term language has constructors for inductive types and destructors for coinductive types. For example, κ_1, κ_2 (to be seen as coprojections) are constructors for the coproduct, and dually, π_1, π_2 (projections) are destructors of the product. The constructor α should be thought of as the initial algebra $\alpha : A[\mu X.A/X] \to \mu X.A$. For example, considering the type Nat, for a term $n : \mathrm{Nat}$, $\alpha(\kappa_2\, n) : \mathrm{Nat}$ represents the successor $n + 1$. Dually, we see the destructor ξ as the structure map of the final coalgebra $\xi : \nu X.A \to A[\nu X.A/X]$. For example, if $s : \mathrm{Str}\, A$ then $\xi\, s : A \times \mathrm{Str}\, A$ represents the pair consisting of the head and tail of s. A variable context Γ consists of a sequence of variable declarations $x : A$. A declaration block Σ consists of a sequence of declarations of the form $f : A = D$ which expresses that the symbol f has type A and its behaviour is defined by the declaration body D where D consists of a sequence of copattern equations $q \mapsto t$. Copatterns q are described below. An **rlet**-expression $\mathbf{rlet}\, \Sigma\, \mathbf{in}\, t$ allows us to use declarations from Σ inside the term t. The name "**rlet**" should be read as "recursive let", emphasising that the declarations are mutually recursive within one binding block. Finally, our term language contains a generalised form of λ-abstraction λD. What usually would be written as $\lambda x.t$ is expressed in our language as $\lambda\{\cdot\, x \mapsto t\}$.

As usual, patterns are used to define functions on inductive types by pattern matching on the arguments, e.g., as in Haskell. Copatterns, on the other hand, are used to define results of applying destructors. Syntactically, copatterns are certain contexts (i.e., terms with a hole $\cdot$). Note that inside a copattern q, we can use patterns p to match on function arguments. The typing judgment $\Gamma \vdash_{\mathrm{cop}} q : A \Rightarrow B$ of copatterns means that a context that expects a term of type A and matches q, results in a term of type B. The definition of a symbol by a mutually recursive set D of copattern equations is closely related to *Behavioural Differential Equations* [43] which are used to specify elements of final coalgebras. For example, the rule **(Ty-D)** is used to assign type $\mathrm{Str}\, A$ to the term

$\lambda\{\mathrm{hd}\cdot \mapsto t_1; \mathrm{tl}\cdot \mapsto t_2\}$ for terms $t_1 : A$ and $t_2 : \mathrm{Str}\, A$, using that $\vdash_{\mathrm{cop}} \mathrm{hd}\ \cdot : \mathrm{Str}\, A \Rightarrow A$ and $\vdash_{\mathrm{cop}} \mathrm{tl}\ \cdot : \mathrm{Str}\, A \Rightarrow \mathrm{Str}\, A$, where $\mathrm{hd} = \pi_1\, \xi$ and $\mathrm{tl} = \pi_2\, \xi$. Another example is given by the canonical term of type 1 which we define as $\langle\rangle := \mathbf{rlet}\, f : 1 = \{\xi \cdot \mapsto f\}\, \mathbf{in}\, f$ which represents a single state with a self-loop. Yet another example, using λ-abstraction, is given by $\lambda\{\pi_1 \cdot \mapsto t_1; \pi_2 \cdot \mapsto t_2\}$ which defines the pair (t_1, t_2).

Definition 2.1. Let Var and Sig be countable sets of term variables and signature symbols. We write

- $\Gamma; \Sigma \vdash t : A$ if t is well-formed term of type A in the variable context Γ using the declarations in Σ;
- $\Gamma; \Sigma \vdash_{\mathrm{bdy}} D : A$ if D is a well-formed declaration body of type A in the variable context Γ using the declarations in Σ;
- $\Sigma_1 \vdash_{\mathrm{dec}} \Sigma_2$ if the declaration block Σ_2 is well-formed using Σ_1;
- $\Gamma \vdash_{\mathrm{pat}} p : A$ if p is a well-formed pattern on the type A that binds variables in Γ;
- $\Gamma \vdash_{\mathrm{cop}} q : A \Rightarrow B$ if q is a well-formed copattern that binds variables in Γ such that if a context e matches q and applying e to a term of type A results in a term of type B.

Well-formed, well-typed terms, and declaration bodies and declaration blocks are given by the rules in Fig. 1 together with the usual weakening, contraction and exchange rules. Well-formed and well-typed (co)patterns are given in Fig. 2.

$$\frac{x \in \mathrm{Var}}{\Gamma, x : A; \Sigma \vdash x : A} \qquad \frac{f \in \mathrm{Sig}}{\Gamma; \Sigma, (f : A = D) \vdash f : A}$$

$$(i = 1, 2)\ \frac{\Gamma; \Sigma \vdash t : A_i}{\Gamma; \Sigma \vdash \kappa_i\, t : A_1 + A_2} \qquad \frac{\Gamma; \Sigma \vdash t : A[\mu X.A/X]}{\Gamma; \Sigma \vdash \alpha\, t : \mu X.A}$$

$$(i = 1, 2)\ \frac{\Gamma; \Sigma \vdash t : A_1 \times A_2}{\Gamma; \Sigma \vdash \pi_i\, t : A_i} \qquad \frac{\Gamma; \Sigma \vdash t : \nu X.A}{\Gamma; \Sigma \vdash \xi\ t : A[\nu X.A/X]}$$

$$\frac{\Gamma; \Sigma \vdash t_1 : A \to B \qquad \Gamma; \Sigma \vdash t_2 : A}{\Gamma; \Sigma \vdash t_1\, t_2 : B}$$

$$\frac{\Gamma; \Sigma \vdash_{\mathrm{bdy}} D : A}{\Gamma; \Sigma \vdash \lambda D : A} \qquad \frac{\Gamma_i \vdash_{\mathrm{cop}} q_i : B \Rightarrow A_i \quad \Gamma, \Gamma_i; \Sigma \vdash t_i : A_i \quad 1 \leq i \leq n}{\Gamma; \Sigma \vdash_{\mathrm{bdy}} \{q_1 \mapsto t_1; \dots; q_n \mapsto t_n\} : B}\ \textbf{(Ty-D)}$$

$$\frac{\emptyset; \Sigma_1, \Sigma_2 \vdash_{\mathrm{bdy}} D : A \qquad \text{all } (f : A = D) \in \Sigma_2}{\Sigma_1 \vdash_{\mathrm{dec}} \Sigma_2} \qquad \frac{\Sigma_1 \vdash_{\mathrm{dec}} \Sigma_2 \qquad \Gamma; \Sigma_1, \Sigma_2 \vdash t : A}{\Gamma; \Sigma_1 \vdash \mathbf{rlet}\, \Sigma_2\, \mathbf{in}\, t : A}$$

Figure 1: Well-formed terms and declaration blocks

Often, we will simplify the notation and write definitions in a Haskell-like style, in that we separate type declaration from the declaration body, and replace the hole $\cdot$ by the symbol we define.

$$\frac{}{x : A \vdash_{\text{pat}} x : A} \qquad \frac{\Gamma \vdash_{\text{pat}} p : A_i}{\Gamma \vdash_{\text{pat}} \kappa_i \, p : A_1 + A_2} \qquad \frac{\Gamma \vdash_{\text{pat}} p : A[\mu X.A/X]}{\Gamma \vdash_{\text{pat}} \alpha \, p : \mu X.A}$$

$$\frac{}{\emptyset \vdash_{\text{cop}} \cdot : A \Rightarrow A} \qquad \frac{\Gamma_1 \vdash_{\text{cop}} q : A \Rightarrow (B \to C) \qquad \Gamma_2 \vdash_{\text{pat}} p : B}{\Gamma_1, \Gamma_2 \vdash_{\text{cop}} q \, p : A \Rightarrow C}$$

$$\frac{\Gamma \vdash_{\text{cop}} q : A \Rightarrow A_1 \times A_2}{\Gamma \vdash_{\text{cop}} \pi_i \, q : A \Rightarrow A_i} \qquad \frac{\Gamma \vdash_{\text{cop}} q : A \Rightarrow \nu X.B}{\Gamma \vdash_{\text{cop}} \xi \, q : A \Rightarrow B[\nu X.B/X]}$$

Figure 2: Well-formed patterns and copatterns

Notation 2.2. We will use the following abbreviations:

Types

$$\begin{aligned} 0 &:= \mu X.X \\ 1 &:= \nu X.X \\ \text{Bool} &:= 1 + 1 \end{aligned}$$

Terms

$$\begin{aligned} \langle\rangle &:= \textbf{rlet}\, f : 1 = \{\xi \cdot \mapsto f\} \,\textbf{in}\, f \\ \text{tt} &:= \kappa_1 \, \langle\rangle & \text{ff} &:= \kappa_2 \, \langle\rangle \\ 0 &:= \alpha(\kappa_1 \, \langle\rangle) & n + 1 &:= \alpha(\kappa_2 \, n) \\ \text{hd } t &:= \pi_1(\xi \, t)) & \text{tl } t &:= \pi_2(\xi \, t)) \\ \lambda x.t &:= \lambda\{\cdot \, x \mapsto t\} \end{aligned}$$

(Co)Patterns

$$\begin{aligned} 0 &:= \alpha(\kappa_1 \, \_) & n + 1 &:= \alpha(\kappa_2 \, n) \\ \text{hd } q &:= \pi_1(\xi \, q)) & \text{tl } q &:= \pi_2(\xi \, q)) \end{aligned}$$

Example 2.3. We start with a very simple example. The stream ones $= (1, 1, 1, \ldots .)$ is defined as the unique solution to the following stream differential equation [43]:

$$\text{hd}(x) = 1, \qquad \text{tl}(x) = x.$$

We define the stream ones as the following term:

$$\textbf{rlet}\, f : \text{Str Nat} = \{ \text{ hd} \cdot \mapsto 1; \text{ tl} \cdot \mapsto f \ \} \,\textbf{in}\, f$$

Example 2.4. As an example of a mixed inductive-coinductive definition, we define a function H of type A with $A = \text{Str Nat} \to \text{Str Nat} \to \text{Str Nat}$ that maps streams s and t

to a stream r with $r(n) = s\left(\sum_{i=0}^{n} t(i)\right)$. The formal definition of H is,

$$
\begin{array}{l}
\mathrm{H} := \textbf{rlet} \\
\quad g : A = \{ \cdot\ s\ t \mapsto f\ (\pi_1(\xi\, t))\ s\ t\ \} \\
\quad f : \mathrm{Nat} \to A = \{ \\
\qquad \pi_1(\xi(\cdot\ \ \alpha(\kappa_1\, x) \quad s\ t)) \mapsto (\pi_1(\xi\, s)) \\
\qquad \pi_2(\xi(\cdot\ \ \alpha(\kappa_1\, x) \quad s\ t)) \mapsto g\ s\ (\pi_2(\xi\, t)) \\
\qquad\qquad (\cdot\ \ \alpha(\kappa_2\, n) \quad s\ t) \ \ \mapsto f\ n\ (\pi_2(\xi\, s))\ t \quad \} \\
\textbf{in } g
\end{array}
$$

In the Haskell-like notation, the example becomes

$$
\begin{array}{l}
\mathrm{H} : A \\
\mathrm{H}\, s\, t = f\ (\mathrm{hd}\, t)\ s\ t \\
f : \mathrm{Nat} \to A \\
\mathrm{hd}(f\ 0 \qquad s\ t) = \mathrm{hd}\, s \\
\ \mathrm{tl}(f\ 0 \qquad s\ t) = \mathrm{H}\, s\ (\mathrm{tl}\, t) \\
\qquad f\ (n+1)\ s\ t\ = f\ n\ (\mathrm{tl}\, s)\ t
\end{array}
$$

The combination of patterns and copatterns is demonstrated in the declaration of f which uses pattern matching on the first argument, followed by specifying $\langle \mathrm{hd}, \mathrm{tl} \rangle\,(f\,0\,s\,t)$.

2.3 Reduction Relation

As in [2, 3], the operational semantics of a program is obtained by applying its defining equations D as rewrite rules. More precisely, we define a reduction relation based on contraction of terms in context.

Definition 2.5. *Evaluation contexts* e are defined by the following grammar:

$$e ::= \cdot \mid e\, t \mid \pi_i\, e \mid \xi\ e$$

For a term r we write $e[r]$ for plugging r into the hole $\cdot$ of e, yielding a term. If $e[r]$ is typeable for any r of type A, we say that e is a context on type A.

These evaluation contexts allow us to define when a rewrite rule can fire. Given a copattern q and a substitution σ, we can obtain an evaluation context $q[\sigma]$ by substituting $\sigma(x)$ for a free variable x with unique occurrence in q. We say that a copattern q matches an evaluation context e, if there is a substitution σ with $q[\sigma] = e$. We use matching to define contraction of terms.

Definition 2.6. Given a term $t : A$ and an evaluation context e, we say that the term $e[t]$ *contracts* to a term t' using declarations in Σ, if $e[t] \succ_\Sigma t'$ can be derived using the following rules:

$$\frac{q_i[\sigma] = e \qquad \text{some } q_i \mapsto t_i \in D}{e[\lambda D] \succ_\Sigma t_i[\sigma]}$$

$$\frac{e[\lambda D] \succ_\Sigma t' \qquad f : A = D \in \Sigma}{e[f] \succ_\Sigma t'}$$

Note that contraction only happens at the top-level of a term, and that Σ is only necessary in the second rule, where a symbol is contracted.

Example 2.7. Let $\Sigma = \{g : A_1 = D_1, f : A_2 = D_2\}$ be the declaration block used to define H in Ex. 2.4, and for $u, v : \mathrm{Str}\,\mathrm{Nat}$ let $e_1 = \cdot\, u\, v\, 0$ and $e_2 = \mathrm{hd}\ e_1$ be evaluation contexts applicable to H. Observe that e_1 neither asks for the head nor for the tail of $f\, u\, v\, 0$. This is where the laziness of the reduction relation lies: $e_1[f]$ does not reduce since $e_1[\lambda D_2]$ does not. On the other hand, we have $\mathrm{hd}(\cdot\, s\, t\, 0)[u/s, v/t] = e_2$ and $\mathrm{hd} \cdot s\, t\, 0 \mapsto \mathrm{hd}\, s \in D_2$, thus $e_2[\lambda D_2] \succ_\Sigma \mathrm{hd}\ u$ and $e_2[f] \succ_\Sigma \mathrm{hd}\ u$.

Using contraction, we define a reduction relation $\longrightarrow_\Sigma$ for each Σ, as the union of reduction relations $\longrightarrow^k_\Sigma$, $k \in \mathbb{N}$, where $\longrightarrow^k_\Sigma$ is a relation on terms of **rlet**-nesting depth k. More precisely, as in [2, 3], $\longrightarrow^k_\Sigma$ is the compatible closure of $\succ^k_\Sigma$ outside of declaration blocks, $\succ^0_\Sigma = \succ_\Sigma$, and $\succ^{k+1}_\Sigma$ is defined inductively by

$$\frac{t \longrightarrow^k_{\Sigma_1,\Sigma_2} t'}{\mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,t \succ^{k+1}_{\Sigma_1} \mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,t'}$$

$$\frac{}{\mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,(\alpha\, t) \succ^{k+1}_{\Sigma_1} \alpha(\mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,t)} \qquad \frac{}{\mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,(\kappa_i t) \succ^{k+1}_{\Sigma_1} \kappa_i(\mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,t)}$$

$$\frac{}{\xi(\mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,t) \succ^{k+1}_{\Sigma_1} \mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,(\xi\, t)} \qquad \frac{}{\pi_i(\mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,t) \succ^{k+1}_{\Sigma_1} \mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,(\pi_i t)}$$

$$\frac{}{(\mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,t)\, s \succ^{k+1}_{\Sigma_1} \mathbf{rlet}\,\Sigma_2\,\mathbf{in}\,(t\, s)}$$

Terms of the form $\mathbf{rlet}\,\Sigma\,\mathbf{in}\,t$ without further declarations in t can be translated into the language of [3] in a way that preserves the reduction relation, we therefore inherit that $\longrightarrow^0_\Sigma$ preserves types. Moreover, we get that a reduction step is always possible on non-normalised terms, if they are well-covering, a notion we briefly describe now.

A sequence Q of copatterns *covers* a type A, written $A \mathrel{\lhd\mid} Q$, if the copatterns in Q are exhaustive and non-overlapping, that is, if for every (large enough) evaluation context e there is exactly one copattern in Q that matches e. For example, the three copatterns in Ex. 2.4 cover the type Str Nat. Non-example are the sequence ($\mathrm{hd}\cdot$; $\mathrm{tl}\cdot$; $\mathrm{hd}\cdot$), as these copatterns are overlapping, and ($\mathrm{hd}(\cdot\, 0)$; $\mathrm{tl}(\cdot\, 0)$), as this sequence is not exhaustive. The covering relation is formally introduced in Appendix A, and is based on that in [3, Sec. 5.2].

We call a term t or a declaration block Σ *well-covering*, if for every declaration body $\{q_1 \mapsto t_1; \dots ; q_n \mapsto t_n\}$ of type A in t or Σ, we have $A \mathrel{\lhd\mid} (q_i)_{i=1,\dots,n}$.

Definition 2.8. For a declaration block Σ and a type A, we denote by $\Lambda^\Sigma(A)$ the set of all closed, well-covering terms t such that $\emptyset; \Sigma \vdash t : A$. We denote by Λ^Σ the union of all $\Lambda^\Sigma(A)$ for $A \in \mathrm{Ty}$. If Σ is empty, we omit Σ and simply write $\Lambda(A)$ and Λ.

Using the notion of well-covering, we can ensure confluence, even in the presence of (co)pattern matching. We prove this using the work by Cirstea and Faure [10] which provides conditions on pattern matching algorithms that ensure confluence. Their last condition, called H_2, is the content of following lemma, which we state here as we will need it later in the proof of Prop. 5.12.

Lemma 2.9. *Let A be a type, Q a set of typed copattern with $A \lhd| \, Q$ and e an evaluation context on A. If there is a $q \in Q$ and a σ with $q[\sigma] = e$ and $e \longrightarrow e'$, then there is a σ' with $q[\sigma'] = e$ and $\sigma \longrightarrow \sigma'$, where the reduction relation is lifted to substitutions point-wise.*

Theorem 2.10. *On $\Lambda^{\Sigma}(A)$, $\longrightarrow_{\Sigma}$ is confluent for any well-covering Σ and, in particular, $\longrightarrow$ is confluent on $\Lambda(A)$.*

Proof. See Appendix A. □

We finish this section with some basic facts about convertibility and strong normalisation. We define *convertibility* of terms, as usual, by

$$t_1 \equiv t_2 \quad \text{iff} \quad \exists t_3 . t_1 \twoheadrightarrow t_3 \twoheadleftarrow t_2 .$$

We note that confluence ensures that $\equiv$ is transitive, hence an equivalence relation. A term t is *strongly normalising*, denoted $t \downarrow$, if there is no infinite reduction sequence starting at t. The set of all strongly normalising terms is denoted by $\mathbf{SN}$. Well-covering terms of inductive type in $\mathbf{SN}$ enjoy the crucial property that they can be reduced to (weak) head normal form (WHNF), i.e., to a form with a constructor in head position, see [3]. Concretely, for $t \in \Lambda(A)$ there is a t' with $t \equiv \kappa_i \, t'$ if $A = B_1 + B_2$, and $t \equiv \alpha \, t'$ if $A = \mu X.B$.

3 Observational Equivalence and Observational Normalisation

In this section, we define formally a notion of *observational equivalence* between terms using *test formulae*. Test formulae are typed and defined inductively in a manner closely related to the testing logic considered in [44, Sec. 6.2], and the many-sorted coalgebraic modal logics for polynomial functors studied, e.g., in [26] and [30, Sec. 2.1.2]. Just as predicates on a set X can be seen as functions $X \to 2$, tests on type A are given an interpretation as terms of type $A \to \text{Bool}$. Tests on inductive types inspect terms by matching on the (weak) head normal form, whereas tests on coinductive types can be thought of as modal formulae. In particular, tests on a function type amount to feeding an argument to a function and testing the result. Here, in order to guarantee that tests always yield a result, we must require that the function arguments are strongly normalising in every evaluation context, a property we call *observational normalisation*, and which can be understood as a formal definition of productivity in our language.

Definition 3.1 (Observational normalisation and tests)**.** Let $A \in \text{Ty}$, i.e., A is a closed, strictly positive type. We simultaneously define tests on A, their interpretation as terms of type $A \to \text{Bool}$, and the set of observationally normalising terms $\mathbf{ON}_A$ of type A.

For a type A, we say that φ is a *test (formula) on A* if we can derive $\varphi : A$ with the rules given below. The set of all test formulae on A is denoted by Tests_A.

$$\frac{}{\top : A} \qquad \frac{}{\bot : A} \qquad \frac{\varphi_1 : A_1 \quad \varphi_2 : A_2}{[\varphi_1, \varphi_2] : A_1 + A_2} \qquad \frac{\varphi : A[\mu X.A/X]}{\alpha^{-1} \, \varphi : \mu X.A}$$

$$\frac{\varphi : A[\nu X.A/X]}{[\xi] \, \varphi : \nu X.A} \qquad \frac{\varphi : A_i}{[\pi_i] \, \varphi : A_1 \times A_2} \qquad \frac{\varphi : B \quad v \in \mathbf{ON}_A}{[v] \, \varphi : A \to B}$$

The interpretation of tests on a type A as terms is given by the map $[\![-]\!]_A : \mathrm{Tests}_A \to \Lambda(A \to \mathrm{Bool})$ which is defined inductively as follows.

$$\begin{aligned}
[\![\top]\!]_A &= \lambda x.\mathrm{tt} && \text{for all } A \in \mathrm{Ty} \\
[\![\bot]\!]_A &= \lambda x.\mathrm{ff} && \text{for all } A \in \mathrm{Ty} \\
[\![[\varphi_1, \varphi_2]]\!]_{A_1+A_2} &= \lambda\{\kappa_1\, x \mapsto [\![\varphi_1]\!]_{A_1}(x)\ ;\ \kappa_2\, x \mapsto [\![\varphi_2]\!]_{A_2}(x)\} \\
[\![\alpha^{-1}\, \varphi]\!]_{\mu X.A} &= \lambda(\alpha\, x).[\![\varphi]\!]_{A[\mu X.A/X]}(x) \\
[\![[\xi]\, \varphi]\!]_{\nu X.A} &= \lambda x.[\![\varphi]\!]_{A[\nu X.A/X]}(\xi\, x) \\
[\![[\pi_i]\, \varphi]\!]_{A_1 \times A_2} &= \lambda x.[\![\varphi]\!]_{A_i}(\pi_i x) \\
[\![[v]\, \varphi]\!]_{A \to B} &= \lambda f.[\![\varphi]\!]_B(f\, v)
\end{aligned}$$

Finally, we define the set $\mathbf{ON}_A$ of *observationally normalising* terms of type A by

$$\mathbf{ON}_A = \{t : A \mid \forall \varphi : A.\ ([\![\varphi]\!]\, t) \in \mathbf{SN}\}.$$

We illustrate with an example how observational normalisation should be seen as a formal definition of productivity.

Example 3.2. Let x and even be given by

$$\begin{aligned}
x &: \mathrm{Str}\,\mathrm{Nat} & \mathrm{even} &: \mathrm{Str}\,\mathrm{Nat} \to \mathrm{Str}\,\mathrm{Nat} \\
\mathrm{hd}\, x &= 1 & \mathrm{hd}\,(\mathrm{even}\, s) &= \mathrm{hd}\, s \\
\mathrm{tl}\, x &= \mathrm{even}\, x & \mathrm{tl}\,(\mathrm{even}\, s) &= \mathrm{even}\,(\mathrm{tl}\,(\mathrm{tl}\, s))
\end{aligned}$$

It is known that x is not productive. To see that x is not observationally normalising, we apply the evaluation context $e = \mathrm{hd}(\mathrm{tl}(\mathrm{tl}\,\cdot))$ to x and find that

$$e[x] \longrightarrow \mathrm{hd}\,(\mathrm{tl}(\mathrm{even}\, x)) \longrightarrow \mathrm{hd}\,(\mathrm{even}\,(\mathrm{tl}(\mathrm{tl}\, x))) \longrightarrow \mathrm{hd}\,(\mathrm{tl}(\mathrm{tl}\, x)) = e[x].$$

Hence there is a diverging reduction sequence starting at $e[x]$, thus $x \notin \mathbf{ON}$. □

Due to the mutual dependency between tests and observationally normalising terms, it is perhaps not immediately clear that their definitions are correct. In order to convince the reader, we show that the induction in the definition of tests in Def. 3.1 is well-founded. We define the order $\mathrm{ord}(A)$ of a type A inductively as follows.

$$\begin{aligned}
\mathrm{ord}(X) &= 0 \\
\mathrm{ord}(A_1 \,\square\, A_2) &= \max_{i=1,2} \mathrm{ord}(A_i), && \square \in \{+, \times\} \\
\mathrm{ord}(\rho X.A) &= \mathrm{ord}(A), && \rho \in \{\mu, \nu\} \\
\mathrm{ord}(A \to B) &= \max\{\mathrm{ord}(A) + 1, \mathrm{ord}(B)\}
\end{aligned}$$

The key property of strictly positive types is that the order of a fixed point type does not increase when unfolding.

Lemma 3.3. *Let A be a type with a free type variable X that occurs only in strictly positive position. We have:*

$$\mathrm{ord}(A[\rho X.A/X]) \leq \mathrm{ord}(\rho X.A), \quad \rho \in \{\mu, \nu\}.$$

Proof. Let $\mathrm{ord}(A) = n$, and B be a type such that X occurs only strictly positively and $\mathrm{ord}(B) \leq n$. We prove by induction in B that $\mathrm{ord}(B[\rho X.A/X]) \leq n$.

In the base case $B = Y$, we get $\mathrm{ord}(B[\rho X.A/X]) = \mathrm{ord}(\rho X.A) = n$ if $Y = X$ and $\mathrm{ord}(B[\rho X.A/X]) = \mathrm{ord}(B) = 0 \leq n$ if $Y \neq X$.

For the induction step:

- If $B = B_1 \mathbin{\square} B_2$, then $\mathrm{ord}(B_i) \leq \max_{i=1,2} \mathrm{ord}(B_i) = \mathrm{ord}(B) \leq n$ and the induction hypothesis applies. Thus $\mathrm{ord}(B_i[\rho X.A/X]) \leq n$, hence $\mathrm{ord}(B[\rho X.A/X]) = \max_{i=1,2} \mathrm{ord}(B_i[\rho X.A/X]) \leq n$.
- If $B = \rho Y.C$ for $\rho \in \{\mu, \nu\}$, then $\mathrm{ord}(B[\rho X.A/X]) = \mathrm{ord}(C[\rho X.A/X]) \leq n$, since $\mathrm{ord}(C) = \mathrm{ord}(B) \leq n$ and the induction hypothesis applies again.
- The interesting case is $B = C \to D$. Since X occurs only strictly positively, we have that $C[\rho X.A/X] = C$, thus $\mathrm{ord}(C[\rho X.A/X]) = \mathrm{ord}(C) \leq \mathrm{ord}(B) - 1$. By induction, we have that $\mathrm{ord}(D[\rho X.A/X]) = k \leq n$ as in the other cases and, by combining these arguments, we find $\mathrm{ord}(B[\rho X.A/X]) = \max\{\mathrm{ord}(C) + 1, k\} \leq n$.

Thus, by induction, we have $\mathrm{ord}(B[\rho X.A/X]) \leq n$ for any B with $\mathrm{ord}(B) \leq n = \mathrm{ord}(A)$ and X occurring only in strictly positive position. Using $B = A$, we get the desired result. □

The order of types allows us to stratify the inductive definition in Def. 3.1. By definition of ord and Lem. 3.3, we have that tests used in the premises of the rules in Def. 3.1 are on types of order less or equal to that of types used in the conclusion. In particular, for tests on function types $A \to B$, we have that $\mathrm{ord}(A) < \mathrm{ord}(A \to B)$, allowing us to define tests on $A \to B$ since Tests_A and $\mathbf{ON}_A$ are fully defined already. Thus, the induction in Def. 3.1 is well-founded. □

To ease readibility, we use the notation $\mathbf{ON} = \bigcup_{A \in \mathrm{Ty}} \mathbf{ON}_A$. Note that $\mathbf{ON} \subseteq \mathbf{SN}$ by definition, and so all terms in $\mathbf{ON}$ have a WHNF.

Remark 3.4. We could have called terms in **ON** *persistently strongly normalising*, in analogy to the similar notion of typed λ-calculus [7, Sec. 17.2], but we favour the name observationally normalising since it fits our setting better.

Definition 3.5 (Test satisfaction, observational equivalence). Let $t \in \Lambda(A)$. We say that t satisfies a test $\varphi : A$, if $t \vDash_A \varphi$ holds, given as follows.

$$\begin{aligned}
&t \vDash_A \top \text{ always holds} \\
&t \vDash_A \bot \text{ never holds} \\
t \vDash_{A_1 + A_2} [\varphi_1, \varphi_2] \quad &\text{iff} \quad \exists t'.\, t \equiv \kappa_i\, t' \text{ and } t' \vDash_{A_i} \varphi_i \\
t \vDash_{\mu X.B} \alpha^{-1} \varphi \quad &\text{iff} \quad \exists t'.\, t \equiv \alpha\, t' \text{ and} \\
&\qquad t' \vDash_{B[\mu X.B/X]} \varphi \\
t \vDash_{\nu X.B} [\xi]\, \varphi \quad &\text{iff} \quad \xi\, t \vDash_{B[\nu X.B/X]} \varphi \\
t \vDash_{A_1 \times A_2} [\pi_i]\, \varphi \quad &\text{iff} \quad \pi_i\, t \vDash_{A_i} \varphi \\
t \vDash_{B \to C} [v]\, \varphi \quad &\text{iff} \quad t\, v \vDash_C \varphi.
\end{aligned}$$

Two terms $t_1, t_2 \in \Lambda(A)$ are *observationally equivalent*, written $t_1 \equiv^A_{\mathrm{obs}} t_2$, if they satisfy the same tests:

$$t_1 \equiv^A_{\mathrm{obs}} t_2 \text{ iff } \forall \varphi : A.\, t_1 \vDash_A \varphi \Leftrightarrow t_2 \vDash_A \varphi.$$

If $t_i = \mathbf{rlet}\, \Sigma\, \mathbf{in}\, t_i$ for some Σ, then we define

$$\Sigma \vdash t_1 \equiv^A_{\mathrm{obs}} t_2 \text{ iff } t_1 \equiv^A_{\mathrm{obs}} t_2.$$

For a context Γ, we say that a substitution σ is **ON**-Γ-*closing*, if $\text{dom}(\Gamma) \subseteq \text{dom}(\sigma)$ and for every $x \in \text{dom}(\Gamma)$, $\sigma(x) : \Gamma(x)$ is a closed **ON**-term. We denote by $\text{Subst}(\Gamma)$ the set of all **ON**-Γ-closing substitutions. We say that two *open* terms t_1, t_2 with $\Gamma \vdash t_1, t_2 : A$ are observationally equivalent, if for all **ON**-Γ-closing σ we have that $t_1[\sigma] \equiv^A_{\text{obs}} t_2[\sigma]$ and denote this by $\Gamma \vdash t_1 \equiv^A_{\text{obs}} t_2$. Analogously, we define $\mathbf{ON}^\Gamma_A$ to be the set of all terms t such that $t[\sigma] \in \mathbf{ON}_A$ for any **ON**-Γ-closing substitution σ.

In what follows, we will frequently omit the type sub- and superscripts and simply write $\vDash$, $[\![-]\!]$ and $\equiv_{\text{obs}}$ when the typing can be inferred from the context.

Remark 3.6. We have given an interpretation of tests on A as terms of type $A \to \text{Bool}$, this is similar to the notion of *observation* from λ-calculus [7, 3.5], but there observations are *any* terms of function type, whereas we restrict to special terms in order to identify more terms. Moreover, we can restrict to Boolean-valued observations, since one can show that allowing any type B as observation output yields the same notion of observational equivalence. This connection with λ-calculus is one reason for the name "observational equivalence", another is that our notion is an instance of the coalgebraic notion of observable behaviour, as we will see in Sec. 5.

The requirement in Def. 3.1 that arguments to functions are closed and observationally normalising is crucial, since without it we can distinguish terms that should be equated. For example, let $f = \lambda x.\, \alpha\, x$ and $g = \lambda\{\alpha\, x \mapsto x\} : \mu X.A \to A[\mu X.A/X]$, and let Ω be the divergent term $\Omega = \mathbf{rlet}\, \omega : \mu X.A = \{\cdot \mapsto \omega\}\, \mathbf{in}\, \omega$ (of type $\mu X.A$). This term does not have a WHNF, hence $f(g\,\Omega) \equiv \alpha(g\,\Omega) \not\equiv \Omega \equiv \text{id}\;\Omega$. This means that, if we would allow Ω as function argument, the test $[\Omega]\,\top$ would distinguish $f \circ g$ and id, thus $f \circ g \not\equiv_{\text{obs}}$ id. However, we want that $f \colon A[\mu X.A/X] \to \mu X.A$ is an initial algebra and so f has to be the inverse of g modulo observational equivalence. Hence we cannot allow terms like Ω as arguments.

A pair of terms that is, rightfully, distinguished by observational equivalence is $\text{H}\,\text{const}_0$ and $\text{H}\,\text{const}_1$, where const_a is the stream everywhere equal to a. For example, the formula $[\text{const}_0]\,[\text{hd}]\,\varphi_{=1}$, where $\varphi_{=1}$: Nat tests for equality to 1, distinguishes them. On the other hand, proving that two terms are observationally equivalent is typically done by induction on tests.

The following lemma shows that the definition of observational equivalence of open terms makes sense.

Lemma 3.7. *For all $x : A \vdash t_i : B$ with $i = 1, 2$, we have*

$$x : A \vdash t_1 \equiv^B_{\text{obs}} t_2 \Leftrightarrow \lambda x.t_1 \equiv^{A \to B}_{\text{obs}} \lambda x.t_2 \tag{1}$$

The next lemma states a number of fundamental properties that we need for showing that types and terms indeed form a category in the next section.

Lemma 3.8. *Observational equivalence $\equiv_{\text{obs}}$ has the following properties.*

(i) Substitutivity: if $\Sigma \vdash t_1 \equiv^A_{\text{obs}} t_2$ and $\Sigma; x : A \vdash r : B$ with $t_1, t_2 \in \mathbf{ON}_A$ and $r \in \mathbf{ON}^{x:A}_B$, then $\Sigma \vdash r[t_1/x] \equiv^B_{\text{obs}} r[t_2/x]$.

(ii) $\equiv_{\text{obs}}$ is an equivalence relation.

(iii) $\equiv_{\text{obs}}$ is a congruence on closed terms in **ON**.

(iv) If $t_1 \equiv^A_{\text{obs}} t_2$, then for all f with $\vdash f : A \to B$ we have $f\, t_1 \equiv^B_{\text{obs}} f\, t_2$.

(v) $\equiv_{\text{obs}}$ *strictly contains convertibility (i.e.,* $\equiv \; \subset \; \equiv_{\text{obs}}$*).*

(vi) $\equiv_{\text{obs}}$ *implies extensionality for terms of function type: if* $t_1, t_2 : A \to B$ *and* $t_1\, v \equiv_{\text{obs}} t_2\, v$ *for all* $v : A$ *in* **ON***, then* $t_1 \equiv_{\text{obs}} t_2$*.*

(vii) $\equiv_{\text{obs}}$ *contains* η*-equivalence:* $\lambda x.tx \equiv_{\text{obs}} t$*,* $x \not\in \text{fv}(t)$*.*

Most of these properties are straightforward to prove, only substitutivity (i) requires a bit more work. The proof uses the rather technical Lem. 3.9 below, which is formulated more generally than needed in the proof of Lem. 3.8, because its proof then becomes easier and we can reuse it later in Sec. 4.2. To ease the formulation of the lemma, we use *conjunctions of tests*. Given tests $\varphi_1, \varphi_2 : A$, their conjunction $\varphi_1 \wedge \varphi_2$ is satisfied by $t : A$ if $t \vDash \varphi_i$ for both $i = 1$ and $i = 2$.

Lemma 3.9. *Let* $f \in \mathbf{ON}_A^\Gamma$ *and let* $\tau \in \text{Subst}(\Gamma)$*. For all tests* $\varphi : A$ *with* $f[\tau] \vDash \varphi$ *there are conjunctive tests* $\{\psi_x\}_{x \in \text{dom}(\Gamma)}$ *such that*

(i) $\tau(x) \vDash \psi_x$ *for all* $x \in \text{dom}(\Gamma)$ *and*

(ii) for all $\sigma \in \text{Subst}(\Gamma)$*, if* $\sigma(x) \vDash \psi_x$ *for all* $x \in \text{dom}(\Gamma)$*, then* $f[\sigma] \vDash \varphi$*.*

Proof. We only sketch the proof. Since f is in $\mathbf{ON}_A^\Gamma$, $[\![\varphi]\!](f[\tau])$ is strongly normalising, thus there is a finite reduction sequence $[\![\varphi]\!](f[\tau]) \twoheadrightarrow N$ to a normal form. We obtain the family $\{\psi_x\}$ by induction in this reduction sequence. In this induction, two cases have to be distinguished: either $\tau(x)$ is contracted within an evaluation context e, in which case ψ_x is extendend by the modalities given by e, or $\tau(x)$ is used as a function argument in a contraction. In the latter case, we reduce the corresponding function to λD or a symbol g, and extend ψ_x by the pattern of D or g that matched $\tau(x)$. Note that λD and g are similar to what Abramsky [4] calls principal weak head normal forms. □

Proof of Lemma 3.8. (i) To show that $r[t_1/x] \equiv_{\text{obs}} r[t_2/x]$, we show that both terms simultaneously satisfy any test $\varphi : B$.

First, assume that $r[t_1/x] \vDash \varphi$. Lemma 3.9 gives us that for $\tau = [t_1/x]$ there is a test $\psi : A$, using the context $\Gamma = x : A$, such that $t_1 \vDash \psi$ and if $t_2 \vDash \psi$, then $r[t_2/x] \vDash \varphi$. Since $t_1 \equiv_{\text{obs}} t_2$, both terms simultaneously satisfy and thus t_2 satisfies ψ and $r[t_2/x] \vDash \varphi$.

Vice versa, $r[t_2/x] \vDash \varphi$ implies, in the same way, that $r[t_1/x] \vDash \varphi$. Summarising, the terms $r[t_1/x]$ and $r[t_2/x]$ satisfy the same tests, hence are observationally equivalent.

(ii) This follows immediately from $\equiv$ being an equivalence.

(iii) Since we are only interested in closed terms, a context is just a term with a free variable, hence (i) and (ii) apply.

(iv) This is just the combination of (i) and Lem. 3.7.

(v) The inclusion is proved by induction in tests and (ii). It is strict by, e.g., item (vi).

(vi) Trivially by the shape of tests on $A \to B$.

(vii) Let $t \colon A \to B$ be a term with $x \not\in \text{fv}(t)$. For every $v \in \mathbf{ON}_A$, we have $(\lambda x.(t\, x))\, v \equiv t\, v$ and hence by (v) and (vi) the equivalence $\lambda x.(t\, x) \equiv_{\text{obs}} t$ tollows.

Finally, we note that $\equiv_{\mathrm{obs}}$ is not trivial, that is, the equational theory $\equiv_{\mathrm{obs}}$ is *consistent* according to [8, Def. 2.1.30].

Proposition 3.10. *The equational theory* $\equiv_{\mathrm{obs}}$ *is consistent.*

Proof. The terms $\mathrm{tt}, \mathrm{ff} : \mathrm{Bool}$ are not observationally equivalent, as they are distinguishable by the test $[\top, \bot]$. Hence $\equiv_{\mathrm{obs}}$ is consistent. □

4 Semantic Properties of Observational Equivalence

We now investigate the semantic implications of taking observational equivalence as equality on terms.

In the first part, we show that by identifying observationally equivalent function terms, we obtain (co)inductive proof principles for reasoning about programs. These proof principles are organised in a category $\mathbb{T}^{\downarrow}$ in which the objects are types and the arrows are (well-formed), observationally normalising terms of type $A \to B$ modulo observational equivalence. We show that this category has coproducts, is Cartesian closed, and that μ-types are initial algebras and ν-types are final coalgebras. The proof principles are thus given by the unique mapping properties of the corresponding structures. In the last part, we show that tests induce a topology on $\mathbb{T}^{\downarrow}$ which makes functions continuous, and in the the case of streams we recover the standardly used prefix topology.

4.1 Category of Observationally Equivalent Terms

We start by briefly recalling the basic definitions of (co)algebras and the unique mapping properties of initial algebras and final coalgebras. For more details, we refer to [42, 27, 14]. Let $\mathbf{C}$ be a category, and $F \colon \mathbf{C} \to \mathbf{C}$ an endofunctor. An *F-algebra* is a $\mathbf{C}$-arrow $a : FA \to A$ in $\mathbf{C}$. A homomorphism of F-algebras from $a : FA \to A$ to $a' : FA' \to A'$ is a $\mathbf{C}$-arrow $f : A \to A'$ such that $a' \circ F(f) = f \circ a$. An initial F-algebra is an F-algebra $\beta : F(\mu F) \to \mu F$ with the property that for every F-algebra $a : FA \to A$ there is a *unique* homomorphism $\overline{a}$ from β to a. We call $\overline{a}$ the *inductive extension of* a.

Dually, an *F-coalgebra* is a $\mathbf{C}$-arrow $c : X \to FX$, and a homomophism of F-coalgebras from $c : X \to FX$ to $c' : X' \to FX'$ is a $\mathbf{C}$-arrow $f : X \to X'$ such that $f \circ c = c' \circ T(f)$. A final F-coalgebra $\omega : \nu F \to F(\nu F)$ is such that for any F-coalgebra $c \colon C \to F(C)$, there is a unique homomorphism $\widetilde{c}$ from c *into* ω, called the *coinductive extension of* c.

These unique mapping properties give rise to proof principles. For initial algebras, we get the familiar induction principle. Dually, the *coinduction principle* can be used to show that two functions h_1, h_2, whose codomain is the carrier of a final coalgebra, are equal by showing that they are homomorphisms, as illustrated in the diagram on the right.

The coalgebra c can be seen as a system of corecursive equations, and the diagram then states that h_1 and h_2 are both solutions to these equations. In our setting, h_1 and h_2 are programs that define some computation. For example, h_1 could be an abstract description and h_2 a concrete implementation. In our category, equality will be observational equivalence, and hence we can use the coinduction principle to show that the implementation is observationally equivalent to the specification.

$$\begin{tikzcd}
X \arrow[r, bend left, "h_1"] \arrow[r, bend right, "h_2"'] \arrow[d, "c"] & \nu F \arrow[d, "\omega"] \\
FX \arrow[r, bend left, "F(h_1)"] \arrow[r, bend right, "F(h_2)"'] & F(\nu F)
\end{tikzcd}$$

We first note that it is possible to define a basic category with types as objects and function terms modulo observational equivalence as arrows. However, this category will not have the desired properties. In order to ensure the existence of coproducts and initial algebras, we must restrict arrows to observationally normalising terms. This restriction is possible since one can easily check that observationally normalising function terms are closed under composition. Moreover, in order to obtain an initial object, we must extend our notion of well-covering to include so-called trivial types, as we explain now.

A good candidate for the initial object is $\mathbf{0} = \mu X.X$ with the term $\lambda\{\} : \mu X.X \to A$ (abstraction without cases) as the unique map for any object A. However, the term $\lambda\{\}$ is not well-covering by the definition of the covering relation in Def. A.1. The reason is that, if we would allow the term $\lambda\{\} : A \to B$ for arbitrary types A, then, given a term $t : A$, the application $(\lambda\{\})\, t$ cannot be reduced to a WHNF and the computation would get stuck. However, if we can ensure that there are not closed terms of type A, then this is not a problem, as the application $(\lambda\{\})\, t$ could never occur. This is captured by the following definition of trivial types.

Definition 4.1. A type A is called *trivial*, if $A = \mu X.X$ or, for trivial types A_1, A_2, C, if $A = A_1 \times A_2$, $A = A_1 + A_2$, $A = B \to C$ or $A = \mu Y.C$.

The notion of a trivial type is known from domain theoretic models for λ-calculus with fixed point types, where the denotation of trivial types is isomorphic to that of 1 [7, Sec. 10.3]. This is reflected by Lem. 4.2.(i).

Lemma 4.2. *Let A be a trivial type, then*

(i) $|\Lambda(A)/{\equiv_{\text{obs}}}| = 1$, *and* *(ii)* $\Lambda(A) \cap \mathbf{ON} = \emptyset$.

The second part of this lemma gives us that there are no observationally normalising terms of trivial type. Hence, since we restrict ourselves in the following to terms in **ON**, we can safely allow the term $\lambda\{\}$ without losing the existence of WHNFs, and thus extend the covering relation from Sec. 2 such that $A \lhd| \emptyset$ whenever A is a trivial type. With this definition, $\lambda\{\} : \mu X.X \to A$ is well-covering.

We can now finally define our category of interest.

Definition 4.3. Let $\overline{\Lambda}(A)$ denote the set of all closed terms of type A that are well-covering with respect to the extended covering relation. For $A \in \text{Ty}$, we let

$$\mathcal{T}(A) = (\overline{\Lambda}(A) \cap \mathbf{ON})/{\equiv_{\text{obs}}}.$$

We define $\mathbb{T}^{\downarrow}$ to be the category in which objects are types from Ty and arrows are given by $\text{Hom}_{\mathbb{T}^{\downarrow}}(A, B) = \mathcal{T}(A \to B)$. The identity arrows are the equivalence classes of $\text{id}_A = \lambda x.x$ and composition is given by composition of representatives: $g \circ f = \lambda x.g(f x)$.

Note that we can find a WHNF for terms in $\overline{\Lambda}(A) \cap \mathbf{ON}$, since, by Lem. 4.2.(ii), there is no term in **ON** to which $\lambda\{\}$ can be applied. Also, note that composition is well-defined due to Lem. 3.7.

Notation 4.4. In what follows, we will implicitly pick representatives for the observational equivalence classes in $\mathcal{T}(A)$. For example, when we say that a term $t \colon C \to D$ is an arrow in $\mathbb{T}^{\downarrow}$, we mean the observational equivalence class of which t is a representative.

In the rest of the section we show that $\mathbb{T}^{\downarrow}$ has the desired properties.

Theorem 4.5. *The category $\mathbb{T}^{\downarrow}$ has all finite coproducts and is Cartesian closed.*

Proof. The final object in $\mathbb{T}^{\downarrow}$ is given by the type $\mathbf{1} = \nu X.X$ and for any type A, the final arrow $!_A \colon A \to \mathbf{1}$ is $!_A = \lambda x.\langle\rangle$. Uniqueness is ensured as all terms of type $\mathbf{1}$ are observationally equivalent to $\langle\rangle$.

The binary product of types A_1, A_2 is the type $A_1 \times A_2$ together with the projections $\pi_i \colon A_1 \times A_2 \to A_i$, $i = 1, 2$. If we are given arrows $f_i \colon B \to A_i$ for $i = 1, 2$ we define the product arrow $\langle f_1, f_2\rangle \colon B \to A_1 \times A_2$ to be

$$\langle f_1, f_2\rangle = \lambda x.\lambda\{\pi_1 \cdot \mapsto f_1\, x \;;\; \pi_2 \cdot \mapsto f_2\, x\}.$$

This arrow clearly fulfils $\pi_i \circ \langle f_1, f_2\rangle \equiv \lambda x.\pi_i(\langle f_1, f_2\rangle\, x) \equiv \lambda x.(f_i\, x) \equiv_{\text{obs}} f_i$. Moreover, it is unique with this property modulo observational equivalence, i.e., if $f \colon B \to A_1 \times A_2$ is such that $\pi_i \circ f \equiv_{\text{obs}} f_i$ then $f \equiv_{\text{obs}} \langle f_1, f_2\rangle$:

$$\begin{aligned}
f &\equiv_{\text{obs}} \lambda x.f\, x \\
&\equiv_{\text{obs}} \lambda x.\lambda\{\pi_1 \cdot \mapsto \pi_1(f\, x) \;;\; \pi_2 \cdot \mapsto \pi_2(f\, x)\} \\
&\equiv_{\text{obs}} \lambda x.\lambda\{\pi_1 \cdot \mapsto (\pi_1 \circ f)\, x \;;\; \pi_2 \cdot \mapsto (\pi_2 \circ f)\, x\} \\
&\equiv_{\text{obs}} \lambda x.\lambda\{\pi_1 \cdot \mapsto f_1\, x \;;\; \pi_2 \cdot \mapsto f_2\, x\} \\
&= \quad \langle f_1, f_2\rangle
\end{aligned}$$

We take $\mathbf{0} = \mu X.X$ as initial object for any type A, the initial arrow $!_A \colon \mathbf{0} \to A$ is $!_A = \lambda\{\}$. Since there are no observationally normalising terms of type $\mathbf{0}$ (by Lem. 4.2.ii), the only tests on $\mathbf{0} \to A$ are $\top$ and $\bot$, hence all terms of type $\mathbf{0} \to A$ are observationally equivalent, and thus $!_A$ is unique.

Next, the binary coproduct of A_1, A_2 is given by $A_1 + A_2$ with inclusions $\kappa_i : A_i \to A_1 + A_2$, $i = 1, 2$. For terms $f_i \colon A_i \to B$ we can form the case distinction $[f_1, f_2] : A_1 + A_2 \to B$ by $[f_1, f_2] = \lambda\{\kappa_1\, x \mapsto f_1\, x \;;\; \kappa_2\, x \mapsto f_2\, x\}$ which factors through the inclusions: $[f_1, f_2] \circ \kappa_i \equiv_{\text{obs}} f_i$. By normalisation we find that $[f_1, f_2]$ is again unique with this property. To this end, let $O \in \text{Obs}(A_1 + A_2 \to B)$ be any observation, then

$$\begin{aligned}
O\, f &\equiv O'(f\, u) && O' \in \text{Obs}(B), u : A_1 + A_2 \text{ closed}, u \in \mathbf{ON} \\
&\equiv O'(f(\kappa_i u')) && u' : A_i \text{ by normalisation} \\
&\equiv O'((f \circ \kappa_i) u') && \lambda x.O'(x\, u') \in \text{Obs}(A_i \to B) \\
&\equiv O'(f_i\, u') \\
&\equiv O'(([f_1, f_2] \circ \kappa_i)\, u') \\
&\equiv O'([f_1, f_2]\; u) \\
&\equiv O\, [f_1, f_2]
\end{aligned}$$

So $f \equiv_{\text{obs}} [f_1, f_2]$ and hence $A_1 + A_2$ is the coproduct of A_1, A_2.

Finally, we show that $\mathbb{T}^{\downarrow}$ is Cartesian closed. The exponential functor $(-)^B : \mathbb{T}^{\downarrow} \to \mathbb{T}^{\downarrow}$ is, as expected, given by $C^B = B \to C$ and $t^B = \lambda f.f \circ t$. We need to show $(-) \times B \dashv (-)^B$, which we do by giving a natural isomorphism $\rho : \text{Hom}((-) \times B, (-)) \xrightarrow{\cong} \text{Hom}((-), (-)^B)$. We define $\rho_{A,C} : \text{Hom}(A \times B, C) \to \text{Hom}(A, C^B)$ to be $\rho_{A,C}(f) = \lambda a.\lambda b.f\,(a, b)$, and its inverse to be $\rho^{-1}_{A,C}(g) = \lambda x.g\,(\pi_1\, x)\,(\pi_2\, x)$.

These are well-defined mappings in the sense that $f \equiv_{\text{obs}} f'$ implies that $\rho_{A,C}(f) \equiv_{\text{obs}} \rho_{A,C}(f')$ and the same for ρ^{-1}, hence these maps respect equivalence classes.

Naturality of ρ is given as follows. Let $t_1 : A' \to A$ and $t_2 : C \to C'$, we need to show that $\text{Hom}(t_1 \times B, t_2)(\rho_{A,C}(f)) = \rho_{A',C'}(\text{Hom}(t_1, t_2^B)(f))$ for all $f : A \times B \to C$. By definition, we get

$$\begin{aligned}
\mathrm{Hom}(t_1 \times B, t_2)(\rho_{A,C}(f)) &= \mathrm{Hom}(t_1 \times B, t_2)(\lambda a.\lambda b.f\,(a,b)) \\
&= (\lambda h.t_2 \circ h) \circ (\lambda a.\lambda b.f\,(a,b)) \circ t_1 \\
&\equiv \lambda x.\big((\lambda h.t_2 \circ h)((\lambda a.\lambda b.f\,(a,b))(t_1\,x))\big) \\
&\equiv \lambda x.\big((\lambda h.t_2 \circ h)(\lambda b.f\,(t_1\,x,b))\big) \\
&\equiv \lambda x.t_2 \circ (\lambda b.f\,(t_1\,x,b)) \\
&\equiv \lambda x.\lambda y.t_2\,(f\,(t_1\,x,y))
\end{aligned}$$

and

$$\begin{aligned}
\rho_{A',C'}(\mathrm{Hom}(t_1, t_2^B)(f)) &= \rho_{A',C'}(t_2 \circ f \circ (t_1 \times \mathrm{id}_B)) \\
&= \lambda a.\lambda b.(t_2 \circ f \circ (t_1 \times \mathrm{id}_B))\,(a,b) \\
&\equiv \lambda a.\lambda b.t_2(f\,(t_1\,a,b))
\end{aligned}$$

which are the same modulo renaming. Thus ρ is a natural transformation.

It remains to be proved that ρ and ρ^{-1} are indeed inverses of each other. The direction $\rho_{A,C} \circ \rho^{-1}_{A,C} = \mathrm{id}$ is easy:

$$\begin{aligned}
\rho_{A,C}(\rho^{-1}_{A,C}(g)) &= \lambda a.\lambda b.\rho^{-1}_{A,C}(g)(a,b) \\
&\equiv \lambda a.\lambda b.g\,(\pi_1\,(a,b))\,(\pi_2\,(a,b)) \\
&\equiv \lambda a.\lambda b.g\,a\,b \\
&\equiv_{\mathrm{obs}} g
\end{aligned}$$

Showing $\rho^{-1}_{A,C} \circ \rho_{A,C} = \mathrm{id}$ is slightly more complicated. First, we notice that for all $u \in \mathbf{ON}_{A\times B}$ we have that $(\pi_1\,u, \pi_2\,u) \equiv_{\mathrm{obs}} u$. This implies that for all $f \in \mathbf{ON}_{A\times B \to C}$ we get $f\,(\pi_1\,u, \pi_2\,u) \equiv_{\mathrm{obs}} f\,u$ by substitutivity (Lemma 3.8), which in turn implies that $\lambda x.f\,(\pi_1\,x, \pi_2\,x) \equiv_{\mathrm{obs}} f$. This gives us

$$\begin{aligned}
\rho^{-1}_{A,C}(\rho_{A,C}(f)) &= \lambda x.\rho_{A,C}(f)\,(\pi_1\,x)\,(\pi_2\,x) \\
&\equiv \lambda x.f\,(\pi_1\,x)\,(\pi_2\,x) \\
&\equiv_{\mathrm{obs}} f
\end{aligned}$$

by the above discussion. Thus $\rho^{-1}_{A,C} \circ \rho_{A,C} = \mathrm{id}$ and each $\rho_{A,C}$ is an isomorphism.

Since ρ^{-1} is the inverse of ρ, it is natural as well, thus $(-) \times B$ is left-adjoint to $(-)^B$ and $\mathbb{T}^{\downarrow}$ is Cartesian closed. □

Next we prove that we can define for each type A, satisfying certain conditions, a functor F_A on $\mathbb{T}^{\downarrow}$, and that fixed point types are initial algebras or final coalgebras for such functors. These two results are proved my mutual induction, since initial algebras and final coalgebras are used to define functors from fixed point types, and conversely, functoriality is needed to obtain initial algebras and final coalgebras for smaller types.

First, we define F_A on objects, which is independent of the mutual induction.

Definition 4.6 (Functors from types, on objects)**.** For a type A in a context $X_1, \dots, X_n$ we define a map on objects of product categories

$$F_A \colon (\mathbb{T}^{\downarrow})^n \to \mathbb{T}^{\downarrow} \quad \text{by substitution} \quad F_A(B_1, \dots, B_n) = A[B_1/X_1, \dots, B_n/X_n].$$

Recall that arrows $(C_1, \dots, C_n) \to (D_1, \dots, D_n)$ in a product category are tuples $(t_1, \dots, t_n)$ with $t_i \colon C_i \to D_i$. We denote such a tuple by $\overrightarrow{t}$ and, analogously, we denote objects by $\overrightarrow{C} = (C_1, \dots, C_n)$. If $n = k + 1$ and $\overrightarrow{C} = (C_1, \dots, C_k)$, then we denote by $F_A^{\overrightarrow{C}} : \mathbb{T}^{\downarrow} \to \mathbb{T}^{\downarrow}$ the mapping $F_A^{\overrightarrow{C}}(D) = F_A\left(\overrightarrow{C}, D\right) = F_{A[\overrightarrow{C}/\overrightarrow{X}]}(D)$. Moreover, we use the following notation:

$$
\begin{aligned}
\mu F_A^{\overrightarrow{C}} &= \mu X.A[\overrightarrow{C}/\overrightarrow{X}], & \alpha_{F_A^{\overrightarrow{C}}} &= \lambda x.\, \alpha\, x : F_A^{\overrightarrow{C}}\left(\mu F_A^{\overrightarrow{C}}\right) \to \mu F_A^{\overrightarrow{C}}, \\
\nu F_A^{\overrightarrow{C}} &= \nu X.A[\overrightarrow{C}/\overrightarrow{X}], & \xi_{F_A^{\overrightarrow{C}}} &= \lambda x.\, \xi\, x : \nu F_A^{\overrightarrow{C}} \to F_A^{\overrightarrow{C}}\left(\nu F_A^{\overrightarrow{C}}\right).
\end{aligned}
$$

Finally, if $k = 0$, then we drop the superscript $\overrightarrow{C}$ and simply write F_A.

We start the mutual induction by defining the action of F_A on arrows.

Lemma 4.7 (Functors from types, on arrows)**.** *For A a type in a context $X_1, \dots, X_n$, we define F_A on an arrow $\overrightarrow{t} : \overrightarrow{C} \to \overrightarrow{D}$ inductively as follows.*

$$
\begin{aligned}
&F_{X_i}(t_1, \dots, t_n) = t_i \\
&F_{\mu X.A}\left(\overrightarrow{t}\right) = \left(\alpha_{F_A^{\overrightarrow{D}}} \circ F_A\left(\overrightarrow{t}, \mathrm{id}_{\mu F_A^{\overrightarrow{D}}}\right)\right)^{-} \\
&F_{A+B}\left(\overrightarrow{t}\right) = F_A\left(\overrightarrow{t}\right) + F_B\left(\overrightarrow{t}\right) \\
&F_{\nu X.A}\left(\overrightarrow{t}\right) = \left(F_A\left(\overrightarrow{t}, \mathrm{id}_{\nu F_A^{\overrightarrow{C}}}\right) \circ \xi_{F_A^{\overrightarrow{C}}}\right)^{\sim} \\
&F_{A \times B}\left(\overrightarrow{t}\right) = F_A\left(\overrightarrow{t}\right) \times F_B\left(\overrightarrow{t}\right) \\
&F_{A \to B}(\overrightarrow{t})(f) = F_B(\overrightarrow{t}) \circ f
\end{aligned}
$$

The bar and tilde superscripts denote inductive and coinductive extensions with respect to $\alpha_{F_A^{\overrightarrow{C}}}$ and $\xi_{F_A^{\overrightarrow{D}}}$.

Proof. We proceed by induction in A. In the base case $A = X_i$, we note that F_{X_i} is the ith projection from the product category, hence a functor.

To prove the induction step, Thm. 4.5 is used to show the functor laws for the compound types $A + B$, $A \times B$ and $A \to B$. The case for the fixed point types was essentially provd in e.g. [28]. We provide the details for convenience. Consider the fixed point type $\nu X.A$. By induction hypothesis, $F_A : (\mathbb{T}^{\downarrow})^n \to \mathbb{T}^{\downarrow}$ is a functor. By Lem. 4.9 (see below), the functors $F_A^{\overrightarrow{C}}$ and $F_A^{\overrightarrow{D}}$ have as final coalgebras $\xi_{F_A^{\overrightarrow{C}}}$ and $\xi_{F_A^{\overrightarrow{D}}}$, and we take $F_{\nu X.A}(\overrightarrow{t})$ to be the coinductive extension of

$$
\nu F_A^{\overrightarrow{C}} \xrightarrow{\xi_{F_A^{\overrightarrow{C}}}} F_A(\overrightarrow{C}, \nu F_A^{\overrightarrow{C}}) \xrightarrow{F_A(\overrightarrow{t}, \mathrm{id})} F_A(\overrightarrow{D}, \nu F_A^{\overrightarrow{C}}).
$$

The functor laws follow from uniqueness, see e.g. [28]. The case $A = \mu X.A$ is treated analogously. □

Remark 4.8. Def. 4.6 and Lem. 4.7 can be proved more generally for types A in which variables X_i occur in either negative position or in positive position. However, since we restrict to strictly positive types the current formulation suffices.

The following lemma is needed in the induction step for fixed points in the proof of Lem. 4.7.

Lemma 4.9. *For any type A with a single free variable X, if the associated F_A defined in Def. 4.6 and Lem. 4.7 is a functor, then α_{F_A} is the initial algebra and ξ_{F_A} the final coalgebra of F_A.*

Proof. Clearly, the algebra and coalgebra structures have the correct types $F_A(\mu X.A) \to \mu X.A$ and $\nu X.A \to F_A(\nu X.A)$, respectively. Well-covering and observational normalisation are evident, as well. For terms $a \colon F_A C \to C$ and $c \colon C \to F_A C$, we define the terms

$$\begin{aligned}&\overline{a} \colon \mu F_A \to C \\ &\overline{a}\,(\alpha\, x) = (a \circ F_A \overline{a})\, x\end{aligned} \qquad \text{and} \qquad \begin{aligned}&\widetilde{c} \colon C \to \nu F_A \\ &\xi\,(\widetilde{c}\, x) = (F_A \widetilde{c} \circ c)\, x.\end{aligned}$$

Clearly, both $\overline{a}$ and $\widetilde{c}$ are closed, typeable and well-covering. With a bit of effort, one can prove that they are also in **ON**, hence they represent arrows in $\mathbb{T}^{\downarrow}$.

We must show that $\overline{a}$ and $\widetilde{c}$ are well-defined, i.e., that they are independent of the choice of representative, and that they are in fact (co)inductive extensions. We achieve this by proving that the terms $\overline{a}$ and $\widetilde{c}$ fulfil the homomorphism equations, and then applying Lem. 4.10 (below).

First, $\widetilde{c}$ is an F_A-coalgebra homomorphism since

$$\begin{aligned}\xi_{F_A} \circ \widetilde{c} \;&\equiv\; \lambda x.\, \xi(\widetilde{c}\, x) \;\equiv\; \lambda x.(F_A \widetilde{c} \circ c)\, x \\ &\equiv\; \lambda x.(F_A \widetilde{c})(c\, x) \;\equiv\; F_A \widetilde{c} \circ c.\end{aligned}$$

For any other representative c' with $c' \equiv_{\text{obs}} c$, we get that $\widetilde{c'}$ is a homomorphism as well, which we use to show that $\widetilde{c'} \equiv_{\text{obs}} \widetilde{c}$ as follows. For any non-trivial test $\psi : C \to \nu X.A$, we must have that $\psi = [v]\,[\xi]\,\psi'$ for some $v : C$ and test $\psi' \colon F_A(\nu X.A)$. Applying Lem. 4.10.1 to c and the test $\varphi = [v]\,\psi' \colon C \to F_A(\nu X.A)$, we obtain a term $t : C \to F_A(\nu X.A)$ such that $t \vDash \varphi \Leftrightarrow \xi_{F_A} \circ \widetilde{c} \vDash \varphi \Leftrightarrow \widetilde{c} \vDash \psi$ and $t \vDash \varphi \Leftrightarrow \xi_{F_A} \circ \widetilde{c'} \vDash \varphi \Leftrightarrow \widetilde{c'} \vDash \psi$. Since this holds for any ψ, we have $\widetilde{c'} \equiv_{\text{obs}} \widetilde{c}$.

In the same way, we show that $\widetilde{c}$ is unique up to observational equivalence, by applying Lem. 4.10.1 to any other F_A-coalgebra homomorphism f from c to ξ_{F_A}. Note that in this case the choice of representative from f does not matter.

Analogously, $\overline{a}$ is well-defined and an F_A-algebra homomorphism by an application of Lem. 4.10.2. So $\overline{a}$ and $\widetilde{c}$ are the unique extensions of a and c, respectively. Hence α_{F_A} is the initial algebra and ξ_{F_A} is the final coalgebra of F_A. □

The following lemma is used in the proof of Lem. 4.9.

Lemma 4.10. *Let A be a type with a single free variable X and assume that F_A as defined in Def. 4.6 and Lem. 4.7 is a functor.*

1. *For every F_A-coalgebra term $c \colon C \to F_A C$ and test $\varphi : C \to F_A(\nu X.A)$, there is a term $t : C \to F_A(\nu X.A)$, such that for any F_A-coalgebra homomorphism $f \colon C \to \nu X.A$ from c to ξ_{F_A}, we have $t \vDash \varphi \Leftrightarrow \xi_{F_A} \circ f \vDash \varphi$.*

2. *For every F_A-algebra term $a \colon F_A C \to C$ and test $\varphi \colon F_A(\mu X.A) \to C$, there is a term $t : F_A(\mu X.A) \to C$, such that for every F_A-algebra homomorphism $f \colon \mu X.A \to C$ from α_{F_A} to a, we have $t \vDash \varphi \Leftrightarrow f \circ \alpha_{F_A} \vDash \varphi$.*

Proof. We only sketch the proof for the first item. Since f is an F_A-coalgebra homomorphism, we have $\xi_{F_A} \circ f \vDash \varphi \Leftrightarrow F_A f \circ c \vDash \varphi$ hence it suffices to prove the existence of a t with $t \vDash \varphi \Leftrightarrow F_A f \circ c \vDash \varphi$. We do this by proving the following claim.

For any subexpression (or ingredient) B of A, any term $c' : C \to F_B C$ and any test $\psi \colon C \to F_B(\nu X.A)$ there is a $t \colon C \to F_B(\nu X.A)$, such that for any F_A-coalgebra homomorphism f from c to ξ_{F_A} the equivalence $t \vDash \psi \Leftrightarrow F_B f \circ c' \vDash \psi$ holds. The result then follows by taking $c' = c$ and $\psi = \varphi$. The claim can be proved by induction in ψ. □

From Lem. 4.7 and Lem. 4.9 the main result of this section follows.

Theorem 4.11. *For all types A with a single free variable, the functor F_A has an initial algebra with carrier $\mu X.A$, and a final coalgebra with carrier $\nu X.A$.*

We demonstrate Lem. 4.7 and Lem. 4.9 on a mixed fixed point type.

Example 4.12. Recall the type $\mathrm{LFair}\, A\, B = \nu X.\mu Y.A \times X + B \times Y$. The type $L = A \times X + B \times Y$ with free variables X, Y induces a functor $F_L : \mathbb{T}^{\downarrow} \times \mathbb{T}^{\downarrow} \to \mathbb{T}^{\downarrow}$ by $F_L(C, D) = A \times C + B \times D$. Since for any type C, the type $\mu Y.A \times C + B \times Y$ is the carrier of an initial algebra of $F_L^C = F_L(C, -)$ (cf. Lem. 4.9), the construct $F_{\mu Y.L}$ as defined in Lem. 4.7 is a functor. Now, we find by Lem. 4.9 that $F_{\mu Y.L}$ has a final coalgebra, whose carrier is $\mathrm{LFair}\, A\, B$.

We illustrate coinduction as proof principle by an elaborate example.

Example 4.13. In Ex. 2.4 we defined a map $\mathrm{H} : \mathrm{Str}\,\mathrm{Nat} \to \mathrm{Str}\,\mathrm{Nat} \to \mathrm{Str}\,\mathrm{Nat}$ of mixed inductive-coinductive type. We also gave a direct definition by explicitly indexing into streams. In this example, we show that both definitions give rise to the same map.

The basic step is to make the coinduction principle applicable. Since streams are a final coalgebra, we redefine H to $\mathrm{H}_1 : \mathrm{Str}\,\mathrm{Nat} \times \mathrm{Str}\,\mathrm{Nat} \to \mathrm{Str}\,\mathrm{Nat}$ by $\mathrm{H}_1\, x = \mathrm{H}\,(\pi_1\, x)\,(\pi_2\, x)$, using that $\mathbb{T}^{\downarrow}$ is Cartesian closed. Now the codomain of H_1 is a final coalgebra and the corresponding coinduction principle is applicable.

Let us give the formal definition of the version of H that uses the explicit indexing. To this end, we define a higher derivative on streams.

$$\begin{aligned}
&\partial : \mathrm{Nat} \to \mathrm{Str}\, A \to \mathrm{Str}\, A \\
&\partial\, 0 \qquad\quad s = s \\
&\partial\, (k+1) \quad s = \partial\, k\, (\mathrm{tl}\, s)
\end{aligned}$$

Moreover, we need a map that sums the first k entries in a stream:

$$\begin{aligned}
&\textstyle\sum_{\leq -} : \mathrm{Nat} \to \mathrm{Str}\,\mathrm{Nat} \to \mathrm{Nat} \\
&\textstyle\sum_{\leq 0} \qquad s = 0 \\
&\textstyle\sum_{\leq k+1} \quad s = (\mathrm{hd}\, s) + \left(\sum_{\leq k} (\mathrm{tl}\, s)\right).
\end{aligned}$$

Using these maps, we can define the alternative version of H with explicit indexing by:

$$\begin{aligned}
&\mathrm{H}_2 : \mathrm{Str}\,\mathrm{Nat} \times \mathrm{Str}\,\mathrm{Nat} \to \mathrm{Str}\,\mathrm{Nat} \\
&\mathrm{H}_2\,(s, t) = \mathrm{toStr}\,(g\, s\, t) \\
&g : \mathrm{Str}\,\mathrm{Nat} \to \mathrm{Str}\,\mathrm{Nat} \to (\mathrm{Nat} \to \mathrm{Nat}) \\
&g\, s\, t\, n = \mathrm{hd}\left(\partial\left(\textstyle\sum_{\leq n+1} t\right)\, s\right)
\end{aligned}$$

This uses one part of the isomorphism $A^{\mathrm{Nat}} \cong \mathrm{Str}\, A$:

$$\begin{aligned}
&\mathrm{toStr} : (\mathrm{Nat} \to A) \to \mathrm{Str}\, A \\
&\mathrm{hd}\,(\mathrm{toStr}\, h) = h\, 0 \\
&\mathrm{tl}\,(\mathrm{toStr}\, h) = \mathrm{toStr}\,(\lambda n.h\,(n+1)).
\end{aligned}$$

With all this set up, we can show that $\mathrm{H}_1 \equiv_{\mathrm{obs}} \mathrm{H}_2$. This is where the universal property of final coalgebras comes into play, since we can define a coalgebra c, such that H_1 and

H_2 are coalgebra homomorphisms from c into the final coalgebra. Hence, by uniqueness, they must be equal.

We use a coalgebra on the type $C = \mathrm{Str\,Nat} \times \mathrm{Str\,Nat}$ given by $c : C \to \mathrm{Nat} \times C$ with $c(s,t) = (\mathrm{hd}\, r, (r, \mathrm{tl}\, t))$ where $r = \partial\,(\mathrm{hd}\, t)\, s$.

First, we show that H_1 is a coalgebra homomorphism. To do this, we need the following intermediate result. For all n, s and t we have that

$$f\,0\;(\partial\, n\, s)\, t \equiv f\, n\, s\, t, \tag{2}$$

which is proved by induction in n. Using this result, we can show $\xi \circ \mathrm{H}_1 \equiv_{\mathrm{obs}} (\mathrm{id} \times \mathrm{H}_1) \circ c$ by a simple calculation. Letting $r = \partial\,(\mathrm{hd}\, t)\, s$, we have

$$\begin{aligned}
(\mathrm{id} \times \mathrm{H}_1)(c(s,t)) &\equiv && (\mathrm{id} \times \mathrm{H}_1)(\mathrm{hd}\, r, (r, \mathrm{tl}\, t)) && \\
&\equiv && (\mathrm{hd}\, r, \mathrm{H}_1\,(r, \mathrm{tl}\, t)) && \\
&\equiv && (\mathrm{hd}\, r, \mathrm{H}\ r\,(\mathrm{tl}\, t)) && \text{Def. } \mathrm{H}_1 \\
&\equiv_{\mathrm{obs}} && \xi(f\,0\, r\, t) && \text{Def. } f \text{ (both cases for 0 combined)} \\
&\equiv && \xi(f\;(\mathrm{hd}\ t)\ s\, t) && \text{by (2)} \\
&\equiv && \xi(\mathrm{H}\, s\, t) && \text{Def. H} \\
&\equiv && \xi(\mathrm{H}_1(s,t)) &&
\end{aligned}$$

Hence, $\xi \circ \mathrm{H}_1 \equiv_{\mathrm{obs}} (\mathrm{id} \times \mathrm{H}_1) \circ c$ by extensionality (Lem. 3.8.(vi)) and H_1 is a homomorphism from c to ξ.

To show that H_2 is a homomorphism as well, we need again two intermediate results:

$$\mathrm{hd}\, r \equiv \quad \mathrm{hd}(\mathrm{H}_2\ s\ t) \tag{3}$$

$$\mathrm{H}_2(r, \mathrm{tl}\, t) \equiv_{\mathrm{obs}} \mathrm{tl}(\mathrm{H}_2\ s\ t) \tag{4}$$

First, we establish the equation for the head of H_2, that is, equation (3).

$$\begin{aligned}
\mathrm{hd}\,(\mathrm{H}_2\, s\, t) &\equiv \mathrm{hd}\,(\mathrm{toStr}(g\, s\, t)) \\
&\equiv g\, s\, t\, 0 \\
&\equiv \mathrm{hd}\left(\partial\,\left(\textstyle\sum_{\leq 1} t\right)\ s\right) \\
&\equiv \mathrm{hd}(\partial\,(\mathrm{hd}\, t)\, s) && \text{by } \mathrm{hd}\, t + 0 \equiv \mathrm{hd}\, t \\
&\equiv \mathrm{hd}\, r
\end{aligned}$$

To establish (4), we need two intermediate results. The first result shows that ∂ is linear in the first argument. It can be proved by induction in n.

$$\partial\,(n+m)\, s \equiv \partial\, m\,(\partial\, n s) \tag{5}$$

The second result is concerned with the evaluation of g at $n+1$.

$$\begin{aligned}
g\, s\, t\,(n+1) &\equiv \mathrm{hd}\left(\partial\,\left(\textstyle\sum_{\leq n+2} t\right)\ s\right) \\
&\equiv \mathrm{hd}\left(\partial\,\left((\mathrm{hd}\, t) + \left(\textstyle\sum_{\leq n+1}(\mathrm{tl}\, t)\right)\right)\ s\right) && \text{Def. of sum} \\
&\equiv \mathrm{hd}\left(\partial\,\left(\textstyle\sum_{\leq n+1}(\mathrm{tl}\, t)\right)(\partial\,(\mathrm{hd}\, t)\, s)\right) && \text{by (5)} \\
&\equiv g\,(\partial\,(\mathrm{hd}\, t)\, s)\,(\mathrm{tl}\, t)\, n
\end{aligned} \tag{6}$$

Using these two equations, we can establish (4).

$$\begin{aligned}
\mathrm{tl}(\mathrm{H}_2\, s\, t) &\equiv && \mathrm{tl}(g\, s\, t) \\
&\equiv && \mathrm{toStr}(\lambda n.(g\, s\, t\,(n+1))) \\
&\equiv && \mathrm{toStr}(\lambda n. g\,(\partial\,(\mathrm{hd}\, t)\, s)\,(\mathrm{tl}\, t)\, n) && \text{by (6)} \\
&\equiv_{\mathrm{obs}} && \mathrm{toStr}(g\,(\partial\,(\mathrm{hd}\, t)\, s)\,(\mathrm{tl}\, t)) && \text{by extensionality} \\
&\equiv && \mathrm{H}_2(\partial\,(\mathrm{hd}\, t)\, s, \mathrm{tl}\, t) \\
&\equiv && \mathrm{H}_2(r, \mathrm{tl}\, t)
\end{aligned}$$

This concludes the intermediate results and we can finally show that H_2 is a homomorphism:

$$\begin{aligned}
(\mathrm{id}\times \mathrm{H}_2)(c(s,t)) &\equiv && (\mathrm{id}\times \mathrm{H}_2)(\mathrm{hd}\, r, (r, \mathrm{tl}\, t)) \\
&\equiv && (\mathrm{hd}\, r, \mathrm{H}_2\,(r, \mathrm{tl}\, t)) \\
&\equiv && (\mathrm{hd}(\mathrm{H}_2\ s\ t), \mathrm{H}_2\,(r, \mathrm{tl}\, t)) && \text{by (3)} \\
&\equiv_{\mathrm{obs}} && (\mathrm{hd}(\mathrm{H}_2\ s\ t), \mathrm{tl}(\mathrm{H}_2\ s\ t)) && \text{by (4)} \\
&\equiv_{\mathrm{obs}} && \xi(\mathrm{H}_2\ s\ t)
\end{aligned}$$

So $\xi \circ \mathrm{H}_2 \equiv_{\mathrm{obs}} (\mathrm{id}\times \mathrm{H}_2) \circ c$ follows from extensionality.

Summarising, we have that both H_1 and H_2 are homomorphisms from c to ξ and thus must be, by uniqueness, observationally equivalent. Hence, up to currying, H is observationally equivalent to H_2, which is given by explicit indexing. □

4.2 Topological Properties

In this section we show that tests induce in a natural way a topology such that arrows in $\mathbb{T}^{\downarrow}$ are continuous. This result relates our work to the constructive view that computable functions are continuous functions, cf. [12]. We also show that on streams and other coinductive types, this topology coincides with the commonly used prefix-induced topology. This tells us that observational equivalence does not identify too much.

Definition 4.14 (Topology on $\mathcal{T}(A)$)**.** Given a term $t \in \mathcal{T}(A)$ and a test $\varphi : A$, we define

$$U_{\varphi}(t) = \{t' \in \mathcal{T}(A) \mid t \vDash \varphi \Leftrightarrow t' \vDash \varphi\}.$$

The topology Θ_A on $\mathcal{T}(A)$ is generated by the subbase

$$\mathcal{U}\,(A) = \{U_{\varphi}(t) \mid \varphi : A, t \in \mathcal{T}(A)\}.$$

Note that the definition of $U_{\varphi}(t)$ is independent of the choice of representative t by the definition of $\equiv_{\mathrm{obs}}$.

We give a topological interpretation of $\mathbb{T}^{\downarrow}$ by defining a functor $F : \mathbb{T}^{\downarrow} \to \mathbf{Top}$ to the category of topological spaces and continuous maps as follows.

$$\begin{array}{ll} \text{for all types } A \in \mathrm{Ty}, & F(A) = (\mathcal{T}(A), \Theta_A) \\ \text{for all } t \in \mathcal{T}(A \to B), & F(t) = \widehat{t} \colon \mathcal{T}(A) \to \mathcal{T}(B) \end{array} \tag{7}$$

where $\widehat{t}$ evaluates t on arguments, that is, for $s \in \mathcal{T}(A)$, $\widehat{t}(s) = t\, s$. Note that $\widehat{t}$ is well-defined due to Lem. 3.7, and F preserves identity and composition since $\mathrm{id}\, s \equiv s$ and $(\widehat{t_2 \circ t_1})(s) \equiv t_2\,(t_1\, s) = \widehat{t_2}(\widehat{t_1}(s))$. It remains to show that for terms t, $\widehat{t}$ is continuous.

Lemma 4.15. *For any $t \in \mathcal{T}(A \to B)$, the map $\widehat{t} \colon \mathcal{T}(A) \to \mathcal{T}(B)$ is continuous with respect to Θ_A and Θ_B.*

Proof. Let $U = U_\varphi(t') \in \mathcal{U}(B)$ be a subbasic open set. We show that for each s in $\widehat{t}^{\leftarrow}(U)$, the $\widehat{t}$-preimage of U, that there is an open set $V_s \subseteq \widehat{t}^{\leftarrow}(U)$ containing s. It follows that $\widehat{t}^{\leftarrow}(U) = \bigcup_s V_s$ is open, hence $\widehat{t}$ is continuous.

We have $s \in \widehat{t}^{\leftarrow}(U)$ iff $t\,s \in U_\varphi(t')$ iff $U = U_\varphi(t\,s)$. By letting $f = t\,x$ we have that $f \in \mathbf{ON}_B^{x:A}$ and $f[s/x] \vDash \varphi$, and we obtain from Lem. 3.9 a conjunctive test $\psi : A$ such that $s \vDash \psi$, and for all $s' \in \mathbf{ON}_A$, $s' \vDash \psi$ implies that $f[s'/x] \vDash \varphi$. In terms of open sets, this implies that $U_\psi(s) \subseteq \widehat{t}^{\leftarrow}(U)$. Furthermore, $U_\psi(s)$ is open since it is the intersection of the subbasic opens $U_{\psi_i}(s)$ for conjuncts ψ_i of ψ, and hence continuity of $\widehat{t}$ follows by taking $V_s = U_\psi(s)$. □

Proposition 4.16. *The map F defined in* (7) *is a functor $F\colon \mathbb{T}^{\downarrow} \to$* **Top**.

We finish the topological investigation by relating the topology Θ_A to classical topologies. To this end, let us, for a test $s : A$ and a type B, denote by $\underline{s}$ application to s, that is, $\underline{s} : (A \to B) \to B$ with $\underline{s} = \lambda f.(f\,s)$. We characterise the topology on Θ_A as "extreme" for the corresponding canonical maps.

Theorem 4.17. *The topologies $\Theta_{A_1 \times A_2}$, $\Theta_{\nu X.A}$ and $\Theta_{A \to B}$ are initial with respect to $\{\widehat{\pi_1}, \widehat{\pi_2}\}$, $\widehat{\xi}$ and $\{\widehat{\underline{s}} \mid s \in \mathbf{ON}_A\}$, respectively (that is, the coarsest topology making these maps continuous). On the other hand, $\Theta_{A_1 + A_2}$ and $\Theta_{\mu X.A}$ are final (i.e., the finest topology) for $\{\widehat{\kappa_1}, \widehat{\kappa_2}\}$ and $\widehat{\alpha}$, respectively.*

As an example, let A be a type such that Θ_A is discrete, for example $A = \mathrm{Nat}$. It is easy to see that in this case the topology on $\mathrm{Str}\,A$ is induced by the usual prefix metric, given by $d(x, y) = 2^{-k}$ with $k = \min\{k \mid \mathrm{hd}\,\mathrm{tl}^k\,x \not\equiv \mathrm{hd}\,\mathrm{tl}^k\,y\}$.

5 Proof Techniques

In this section, we present coinductive methods for proving that two terms are observationally equivalent, and for proving that a term is observationally normalising. We do so by defining a transition system on terms such that observational equivalence coincides with bisimilarity, and hence observational equivalence of two terms can be proved by establishing a bisimulation relation containing them. On the same transition system, we show that observational normalisation is a coinductive predicate such that observational normalisation can be proved by establishing a subset of strongly normalising terms that is closed under transitions. Moreover, we provide a number of up-to techniques that will enhance these proof techniques.

5.1 Terms as Transition System

We define a transition structure in which the successors of a term t are the terms (modulo reductions) that a test can inspect in order to determine whether it is satisfied by t. For example, a term t of type $C = A_1 + A_2$ can either be reduced to a term of the form $\kappa_i\,t'$ for some i and term $t' : A_i$, or t does not have a WHNF. In the first case, the successors of t are all those t' such that $t \twoheadrightarrow \kappa_i\,t'$ (these are the terms that will be inspected by tests). In the second case, no further inspection is possible and there is no outgoing transition from t.

Formally, the transition structure of terms is a coalgebra in the category of families of sets indexed by the types of our type system. Let I be a set, the category $\mathbf{Set}^I$ has as objects families of sets $X = \{X_i\}_{i \in I}$ indexed by I, and as morphisms $f : X \to Y$ families of functions $\{f_i : X_i \to Y_i\}_{i \in I}$.

The branching type is given by a functor $F \colon \mathbf{Set}^{\mathrm{Ty}} \to \mathbf{Set}^{\mathrm{Ty}}$ such that for $C \in \mathrm{Ty}$, $F(X)_C$ consists of sets indexed by the types of subterms that can be inspected by tests on type C. Formally, F is defined by:

$$F(X)_C = \begin{cases} \coprod_{i \in \{1,2\}} \mathcal{P}\left(X_{A_i}\right) + \{*\}, & C = A_1 + A_2 \\ \mathcal{P}\left(X_{A[\mu X.A/X]}\right) + \{*\}, & C = \mu X.A \\ \prod_{i \in \{1,2\}} \mathcal{P}\left(X_{A_i}\right), & C = A_1 \times A_2 \\ \mathcal{P}\left(X_{A[\nu X.A/X]}\right), & C = \nu X.A \\ \mathcal{P}\left(X_B\right)^{\mathbf{ON}_A}, & C = A \to B \end{cases}$$

where $\mathcal{P}(-)$ is the covariant powerset functor, and for a set U we denote by $(-)^U$ the function space functor. F acts on morphisms in the obvious way. The component for coproducts was explained above. The other components can be understood similarly following Def. 3.5.

Let $\Lambda = \{\Lambda(A)\}_{A \in \mathrm{Ty}}$ and define the coalgebra on terms $\delta : \Lambda \to F(\Lambda)$ by

$$\delta_{A_1+A_2}(t) = \begin{cases} \iota_i(\{t' : A_i \mid t \twoheadrightarrow \kappa_i\, t'\}), & \exists t'.\, t \twoheadrightarrow \kappa_i\, t' \\ *, & \text{otherwise} \end{cases}$$

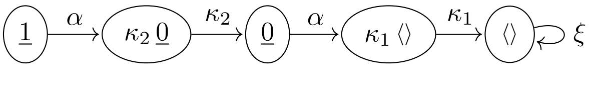
$$\delta_{\mu X.A}(t) = \begin{cases} \{t' : A \mid t \twoheadrightarrow \alpha\ t'\}, & \exists t'.\, t \twoheadrightarrow \alpha\ t' \\ *, & \text{otherwise} \end{cases}$$

$$\begin{aligned} \delta_{A_1 \times A_2}(t)(i) &= \{t' : A_i \mid \pi_i\, t \twoheadrightarrow t'\} \\ \delta_{\nu X.A}(t) &= \{t' : A \mid \ \xi\ t \twoheadrightarrow t'\} \\ \delta_{A \to B}(t)(u) &= \{t' : C \mid t\, u \twoheadrightarrow t'\} \end{aligned}$$

Note that δ is well-defined on terms of sum type by confluence.

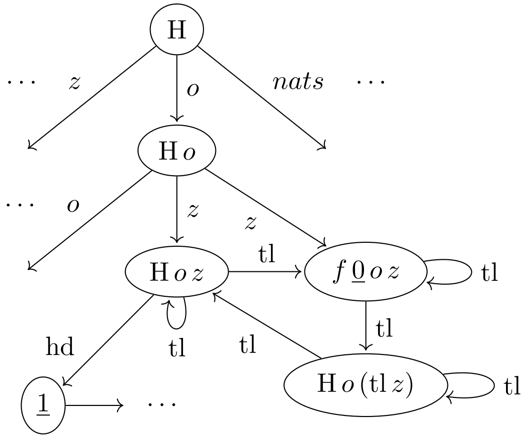
Example 5.1. We give two examples of the structure δ, the first for an inductive and the second on a coinductive type. Since δ is a transition system, we use the common way of displaying such systems by nodes and arrows. A node is labelled by the term it represents. Transition arrows are provided with a label that indicates how the successor was obtained.

1. We denote by $\underline{n}$ the representation of a natural number as term, that is, $\underline{0} = \alpha(\kappa_1 \langle\rangle)$ and $\underline{n+1} = \alpha(\kappa_2\, \underline{n})$. We display below some of the transitions starting in $\underline{1}$.



   Of course, there are also terms of type Nat that do not have a WHNF. For example, let Σ be the definition block $\Sigma = (f : \mathrm{Nat} = \{\cdot \mapsto f\})$, in which case we have $\delta_{\mathrm{Nat}}(\mathbf{rlet}\, \Sigma \,\mathbf{in}\, f) = *$.

2. The second example is H. We denote by $o, z : \mathrm{Str\,Nat}$ the streams that are constantly $\underline{1}$ and $\underline{0}$, respectively. We show a part of δ starting at H where o and z are used as arguments.



Remark 5.2. Note that F only distinguishes between inductive and coinductive types:

$$F(X)_C \cong \begin{cases} \coprod_{i\in\{1,2\}} \mathcal{P}\left(X_{A_i}\right) + \{*\}, & C = A_1 + A_2 \\ \coprod_{i\in\{*\}} \mathcal{P}\left(X_{A[\mu X.A/X]}\right) + \{*\}, & C = \mu X.A \\ \prod_{i\in\{1,2\}} \mathcal{P}\left(X_{A_i}\right), & C = A_1 \times A_2 \\ \prod_{i\in\{*\}} \mathcal{P}\left(X_{A[\nu X.A/X]}\right), & C = \nu X.A \\ \prod_{i\in\mathbf{ON}_A} \mathcal{P}\left(X_B\right), & C = A \to B \end{cases}$$

This is interesting because it makes clear how the duality between inductive and coinductive types arises in the term coalgebra: Observations on inductive types are given by a reduction WHNF, if possible, and removing the constructor in head position. Observations for coinductive types, on the other hand, are just given by applying the destructors of the corresponding type.

5.2 Observational Equivalence as Bisimilarity

In this section, we establish that observational equivalence is a bisimulation. We use the definition of bisimulation in terms of relation lifting, see e.g. [45]. For a family $X \in \mathbf{Set}^{\mathrm{Ty}}$, we denote by $\mathrm{Rel}(X)$ the poset category of relation families $R = \{R_A\}_{A\in\mathrm{Ty}}$ where $R_A \subseteq X_A \times X_A$, and the order is given by: $R \sqsubseteq R'$ iff $R_A \subseteq R'_A$ for each $A \in \mathrm{Ty}$. The lifting of F at Λ is defined as the functor $\overline{F} : \mathrm{Rel}(\Lambda) \to \mathrm{Rel}(F(\Lambda))$ where $\overline{F}(R)$ is the image of the map $F(R) \xrightarrow{\langle F(\pi_1), F(\pi_2)\rangle} F(\Lambda) \times F(\Lambda)$, where $\pi_i \colon R \to \Lambda, i = 1, 2$, are the projections of R. That is, for all $(U_1, U_2) \in F(\Lambda)_A \times F(\Lambda)_A$,

$$(U_1, U_2) \in \overline{F}(R)_A \iff \exists U \in F(R)_A : U_k = (F\pi_k)_A(U).$$

A *bisimulation on* δ is a relation $R \in \mathrm{Rel}(\Lambda)$ such that $R \sqsubseteq \Phi(R)$ where $\Phi(R) = (\delta \times \delta)^{-1}(\overline{F}(R))$, i.e.,

$$\Phi(R)_A = \{(t_1, t_2) \in \Lambda(A)^2 \mid (\delta_A(t_1), \delta_A(t_2)) \in \overline{F}(R)_A\}.$$

We can now formulate the main result of this section.

Theorem 5.3. *The largest bisimulation on* δ *is* $\equiv_{\mathrm{obs}}$.

Proof. To show that $\equiv_{\text{obs}}$ is a bisimulation on δ, we need to show that $\equiv_{\text{obs}} \sqsubseteq \Phi(\equiv_{\text{obs}})$. So we let $t_1 \equiv^A_{\text{obs}} t_2$ for some type A and show that $(t_1, t_2) \in \Phi(\equiv_{\text{obs}})_A$.

- For $A = A_1 + A_2$, we distinguish three cases.

  i) The term t_1 has a WHNF, i.e., there are $i \in \{1, 2\}$ and t'_1 with $t_1 \twoheadrightarrow \kappa_i\, t'_1$. But then there is a t'_2 with $t_2 \twoheadrightarrow \kappa_i\, t'_2$, for otherwise one of the tests $[\top, \bot]$ or $[\bot, \top]$ would distinguish t_1 and t_2, contradicting $t_1 \equiv_{\text{obs}} t_2$. Thus t_2 has a WHNF, too.

     We show that for t'_k with $t_k \twoheadrightarrow \kappa_i\, t'_k$, $t'_1 \equiv_{\text{obs}} t'_2$ must hold. In the case $i = 1$, assume there is a test φ distinguishing t'_1 and t'_2. Then $[\varphi, \bot]$ distinguishes t_1 and t_2, contradicting $t_1 \equiv_{\text{obs}} t_2$. Thus we must have $t'_1 \equiv_{\text{obs}} t'_2$. The case of $i = 2$ is symmetric. We put $U = \{(t'_1, t'_2) \mid t_k \twoheadrightarrow \kappa_i\, t'_k\}$ and by the two arguments above, we have $\iota_i(U) \in F(\equiv_{\text{obs}})_A$ and $\delta(t_k) = \iota_i(\pi_k(X))$, thus $(t_1, t_2) \in \Phi(\equiv_{\text{obs}})_A$.

  ii) Vice versa, if t_2 has a WHNF, $(t_1, t_2) \in \Phi(\equiv_{\text{obs}})_A$ using a symmetric argument.

  iii) If neither t_1 nor t_2 has a WHNF, then $\delta(t_k) = *$ for $k = 1, 2$ and, since $(*, *) \in \overline{F}(\equiv_{\text{obs}})_A$, we have that $(t_1, t_2) \in \Phi(\equiv_{\text{obs}})_A$.

- The case $A = \mu X.B$ is proved analogously.

- If $A = B \to C$, we have for each $u \in \mathbf{ON}_B$ and t'_k with $t_k\, u \twoheadrightarrow t'_k$ that $t'_1 \equiv_{\text{obs}} t'_2$. Since suppose t'_1 and t'_2 could be distinguished by a test φ, then the test $[u]\, \varphi$ would distinguish t_1 and t_2, contradicting $t_1 \equiv_{\text{obs}} t_2$. Thus $t'_1 \equiv_{\text{obs}} t'_2$, for all $u \in \mathbf{ON}_B$ and $t_k\, u \twoheadrightarrow t'_k$.

  Now let $X(u) = \{(t'_1, t'_2) \mid t_k\, u \twoheadrightarrow t'_k\}$. By the above discussion, $X \in \overline{F}(\equiv_{\text{obs}})_A$, and by definition, also $\pi_k(X(u)) = \delta(t_k)(u)$ for all $u \in \mathbf{ON}_B$, hence $(t_1, t_2) \in \Phi(\equiv_{\text{obs}})_A$.

- The cases for products and greatest fixed point types can be proved analogously.

It remains to prove that $\equiv_{\text{obs}}$ is the largest such bisimulation. So let $R \sqsubseteq \Lambda^2$ be such that $R \sqsubseteq \Phi(R)$. We show that $R \sqsubseteq \equiv_{\text{obs}}$ by showing that for all $A \in \text{Ty}$ and all $(t_1, t_2) \in R_A$, $t_1 \vDash \varphi \Leftrightarrow t_2 \vDash \varphi$. The proof is by induction in φ. The base case for the trivial tests $\top$ and $\bot$ is immediate. We prove the induction step by case distinction in A.

- If $A = A_1 + A_2$, then, since R is a bisimulation, either $\delta(t_1) = \delta(t_2) = *$ or there is an $i \in \{1, 2\}$ such that

$$\forall t'_1.\, t_1 \twoheadrightarrow \kappa_i\, t'_1,\ \exists t'_2.\, t_2 \twoheadrightarrow \kappa_i\, t'_2 \wedge (t'_1, t'_2) \in R_{A_i} \tag{8}$$
$$\forall t'_2.\, t_2 \twoheadrightarrow \kappa_i\, t'_2,\ \exists t'_1.\, t_1 \twoheadrightarrow \kappa_i\, t'_1 \wedge (t'_1, t'_2) \in R_{A_i} \tag{9}$$

  We use this to show that t_1 and t_2 satisfy φ simultaneously. If $\delta(t_k) = *$, then t_k does not have a WHNF and both t_1 and t_2 do not satisfy φ. Otherwise, we use that the test φ must be of the form $[\psi_1, \psi_2]$ with $\psi_i : B_i$. By definition, $t_1 \vDash \varphi \Leftrightarrow t'_1 \vDash \psi_i$ for $t_1 \equiv \kappa_i\, t'_1$. The existence of t'_1 with $t_1 \equiv \kappa_i\, t'_1$ implies that there is a t''_1 with $t_1 \twoheadrightarrow \kappa_i\, t''_1$. By (8), there is a t'_2 with $t_2 \twoheadrightarrow \kappa_i\, t'_2$ and $(t''_1, t'_2) \in R_{A_i}$. Finally, by induction hypothesis, t'_1 and t'_2 simultaneously satisfy ψ_i, hence $t_2 \vDash \varphi$. Using (9), we prove analogously that $t_2 \vDash \varphi$ implies $t_1 \vDash \varphi$.

- We proceed analogously for least fixed point types.

- If $A = B \to C$, then the assumption $R_A \subseteq \Phi(R)_A$ says that for all $u \in \mathbf{ON}_B$

  $$\forall t_1'.\, t_1\, u \twoheadrightarrow t_1',\ \exists t_2'.\, t_2\, u \twoheadrightarrow t_2' \ \wedge\ (t_1', t_2') \in R_C \tag{10}$$
  $$\forall t_2'.\, t_2\, u \twoheadrightarrow t_2',\ \exists t_1'.\, t_1\, u \twoheadrightarrow t_1' \ \wedge\ (t_1', t_2') \in R_C \tag{11}$$

  This allows us to show that $t_1 \vDash \varphi \Leftrightarrow t_2 \vDash \varphi$. The test φ must be of the form $[u]\,\psi$ for some $u \in \mathbf{ON}_B$ and $\psi : C$. By (10), there is a $t_2\, u \twoheadrightarrow t_2'$ with $(t_1\, u, t_2') \in R_C$, which implies by induction that $t_1\, u$ and t_2' simultaneously satisfy ψ. Moreover, $t_2\, u \twoheadrightarrow t_2'$ implies that $t_2\, u$ and t_2' simultaneously satisfy ψ. Hence $t_1\, u \vDash \psi \Rightarrow t_2\, u \vDash \psi$ and thus $t_1 \vDash \varphi \Rightarrow t_2 \vDash \varphi$. Analogously, we prove $t_2 \vDash \varphi \Rightarrow t_1 \vDash \varphi$ by (11), thus t_1 and t_2 simultaneously satisfy φ.

- The case for products and greatest fixed points is proved analogously. □

We now have a bisimulation proof principle for observational equivalence. However, bisimulations on δ are very big, since successor sets are closed under reductions. One way of ensuring all necessary pairs are included is to close the bisimulation under convertibility. Moreover, we cannot use equational reasoning with respect to observational equivalence directly, we rather need to close the bisimulation under observational equivalence by hand. If we try to use Thm. 5.3 directly, we will find ourselves constructing very complicated bisimulations. There are several reasons to this. The first is that a bisimulation relation needs to contain *all* pairs of convertible terms, and to achieve this we would usually close the bisimulation, we are interested in, under convertibility. Secondly, we cannot use equational reasoning with respect to observational equivalence directly, we rather need to again close the bisimulation under observational equivalence by hand. Finally, we need to add trivial pairs to achieve, for example, that a relation is reflexive.

Having to explicitly close bisimulations as just described can be avoided by using so-called *(bisimulation) up-to techniques* [9, 38, 39, 41]. Formally, an up-to technique is defined as an endofunctor $T : \mathrm{Rel}(X) \to \mathrm{Rel}(X)$ on relations over a family X. However, not every such functor is useful, rather we are interested in *sound* up-to techniques, that is, up-to techniques T such that $R \sqsubseteq \Phi(T(R))$ implies $R \sqsubseteq \equiv_{\mathrm{obs}}$. $R \sqsubseteq \Phi(T(R))$ implies $R \sqsubseteq \equiv_{\mathrm{obs}}$.

Various up-to techniques for enhancing the bisimulation proof method have been studied in recent years [9, 38, 41]. An important concept that emerged in [38] is that of *compatible* up-to techniques, where T is said to be Φ-compatible, if $T \circ \Phi \sqsubseteq \Phi \circ T$. It has been shown in the abovementioned papers that compatible up-to techniques are sound and that they can be composed. We make use of these facts in the following to establish an up-to technique that solves all the problems of the bisimulation proof principle that we mentioned above.

For $R \in \mathrm{Rel}(\Lambda)$, we denote by $R^{\equiv_{\mathrm{obs}}} \in \mathrm{Rel}(\Lambda)$ the closure of R under observational equivalence, that is,

$$R^{\equiv_{\mathrm{obs}}} \ :=\ \equiv_{\mathrm{obs}} \circ R \circ \equiv_{\mathrm{obs}} .$$

Since $\overline{F}$ is the canonical lifting of F to relations, we get from [9] that up-to bisimilarity, i.e. up-to $\equiv_{\mathrm{obs}}$, is compatible. Furthermore, we denote by Eq the diagonal relation and by $\sqcup$ index-wise union. Then we have the following result.

Proposition 5.4. *The functor T given by $T(R) = (R \sqcup \mathrm{Eq})^{\equiv_{\mathrm{obs}}}$ is a compatible up-to technique. Hence, if $R \sqsubseteq \Phi\left((R \sqcup \mathrm{Eq})^{\equiv_{\mathrm{obs}}}\right)$, then $R \sqsubseteq \equiv_{\mathrm{obs}}$.*

Proof. Since $\mathrm{Eq} \sqsubseteq \Phi(\mathrm{Eq})$, we have that the constant functor mappping to Eq is compatible [9, Prop. 1]. By the same proposition, also $(-)^{\equiv_{\mathrm{obs}}}$ and $\sqcup$ are compatible. Hence the T composition is compatible, and soundness follows. □

We reiterate Example 4.13 to demonstrate the results of this section for proving observational equivalence.

Example 5.5. We claim that the relation

$$\begin{aligned} R_{\mathrm{Str\,Nat} \times \mathrm{Str\,Nat} \to \mathrm{Str\,Nat}} &= \{(H_1, H_2)\} \\ R_{\mathrm{Str\,Nat}} &= \{(H_1\,(s,t), H_2\,(s,t)) \mid s,t \in \mathbf{ON}_{\mathrm{Str\,Nat}}\} \\ R_{\mathrm{Nat} \times \mathrm{Str\,Nat}} &= \{((u, H_1\,(s,t)), (u, H_2\,(s,t))) \mid u \in \mathbf{ON}_{\mathrm{Nat}},\ s,t \in \mathbf{ON}_{\mathrm{Str\,Nat}}\} \\ R_A &= \emptyset, \quad \text{otherwise} \end{aligned}$$

is a bisimulation up to T (defined in Prop. 5.4) hence it proves that $H_1 \equiv_{\mathrm{obs}} H_2$.

Proof. We need to show that $R_A \subseteq \Phi(T(R))_A$ for all types A. The first case of R is easy. If $(s,t) \in \mathbf{ON}_{\mathrm{Str\,Nat} \times \mathrm{Str\,Nat}}$, then we have for all r_1, r_2 with $H_i\,(s,t) \twoheadrightarrow r_i$ that $(r_1, r_2) \in T(R)_{\mathrm{Str\,Nat}}$, as T closes R under observational equivalence and therefore under convertibility. For the second case, recall that we proved in Ex. 4.13 the equations

$$\begin{aligned} &\xi(H_1\,(s,t)) \equiv_{\mathrm{obs}} (\mathrm{hd}\, r, H_1(r, \mathrm{tl}\, t)) \text{ and} \\ &\xi(H_2\,(s,t)) \equiv_{\mathrm{obs}} (\mathrm{hd}\, r, H_2(r, \mathrm{tl}\, t)). \end{aligned}$$

It immediately follows that all r_1, r_2 with $\xi(H_i\,(s,t)) \twoheadrightarrow r_i$ are related by $T(R)_{\mathrm{Nat} \times \mathrm{Str\,Nat}}$. Finally, $R_{\mathrm{Nat} \times \mathrm{Str\,Nat}} \subseteq \Phi(T(R))_{\mathrm{Nat} \times \mathrm{Str\,Nat}}$ follows since $\pi_1(u, H_i\,(s,t))$ reduces to u and $T(R)$ contains the diagonal relation, and because $\pi_2(u, H_i\,(s,t))$ is convertible to $H_i\,(s,t)$. Thus R is a bisimulation up to T and $H_1 \equiv_{\mathrm{obs}} H_2$. □

5.3 Observational Normalisation as Coinductive Predicate

After this warm-up, we show that **ON** is the largest predicate on terms among those that only contain strongly normalising terms and are closed under δ transitions.

The scene for this is set as follows. Given $X \in \mathbf{Set}^{\mathrm{Ty}}$, Pred_X is the poset category that consists of all predicates over X, that is, families $P \in \mathbf{Set}^{\mathrm{Ty}}$ with $P \sqsubseteq X$, ordered by inclusion. As a predicate, **ON** lives in Pred_Λ, and we define $\delta^* : \mathrm{Pred}_{F\Lambda} \to \mathrm{Pred}_\Lambda$, the reindexing along δ, by taking index-wise preimages:

$$\delta^*(P)_A = \delta_A^{-1}(P_A).$$

Functoriality of δ^* is ensured, since taking preimages is a monotone operation. As for bisimilarity, we need a lifting $S : \mathrm{Pred}_\Lambda \to \mathrm{Pred}_{F\Lambda}$ of the functor F, this time to predicates. This lifting is essentially just F restricted to Pred_Λ, but since $\Lambda \subseteq \mathbf{SN}$ we do not need the $*$ as every strongly normalising term has a WHNF, and we define S by,

$$S(P)_C = \begin{cases} \bigcup_{k \in \{1,2\}} \{\iota_k X \mid X \subseteq P_{A_k}\}, & C = A_1 + A_2 \\ \{\iota_1 X \mid X \subseteq P_{A[\mu X.A/X]}\}, & C = \mu X.A \\ F(P)_C, & \text{otherwise} \end{cases}$$

It is straightforward to check that S is a functor and a lifting of F, that is, $P \sqsubseteq Q$ implies $S(P) \sqsubseteq S(Q)$ and $S(P) \sqsubseteq F(\Lambda)$. The final ingredients we need are the constant functor

$K_{\mathbf{SN}} \colon \mathrm{Pred}_\Lambda \to \mathrm{Pred}_\Lambda$ that maps every predicate to $\mathbf{SN}$, and the diagonal $\Delta : \mathrm{Pred}_\Lambda \to \mathrm{Pred}_\Lambda^2$.

We combine all of this into one functor $\Psi : \mathrm{Pred}_\Lambda \to \mathrm{Pred}_\Lambda^2$ by putting $S_\delta = \delta^* \circ S$ and $\Psi = \langle K_{\mathbf{SN}}, S_\delta \rangle$. The purpose of Ψ is to characterise the two properties of observationally normalising terms: these are strongly normalising and an observation results in an observationally normalising term.

For the sake of clarity, we first give an explicit description of S_δ on sums and function spaces.

$$\begin{aligned}
S_\delta(P)_{A_1+A_2} &= \delta^{-1}_{A_1+A_2}(S(P)_{A_1+A_2}) \\
&= \{t : A_1 + A_2 \mid \exists i \in \{1,2\}.\, \exists X \subseteq P_{A_i}.\, \delta(t) = \iota_i X\} \\
&= \{t : A_1 + A_2 \mid \exists i \in \{1,2\}.\, \exists X.\, \delta(t) = \iota_i X \wedge X \subseteq P_{A_i}\} \\
&= \{t : A_1 + A_2 \mid \exists i \in \{1,2\}.\, (\exists t'.\, t \twoheadrightarrow \kappa_i\, t') \wedge (\forall t'.\, t \twoheadrightarrow \kappa_i\, t' \Rightarrow t' \in P_{A_i})\}
\end{aligned}$$

$$\begin{aligned}
S_\delta(P)_{C \to D} &= \delta^{-1}_{C \to D}(F(P)_{C \to D}) \\
&= \{t : C \to D \mid \delta(t) \in \mathcal{P}\,(P_D)^{\mathbf{ON}_C}\} \\
&= \{t : C \to D \mid \forall u \in \mathbf{ON}_C.\, \delta(t)(u) \subseteq P_D\} \\
&= \{t : C \to D \mid \forall u \in \mathbf{ON}_C.\, \forall t'.\, t\, u \twoheadrightarrow t' \Rightarrow t' \in P_D\}
\end{aligned}$$

Using this setup, we can formulate the main result of this section.

Theorem 5.6. *The predicate* $\mathbf{ON}$ *is the largest predicate* P *on* Λ *such that*

$$\Delta(P) \sqsubseteq \Psi(P)$$

where the inclusion is given point-wise in the product category $\mathrm{Pred}_\Lambda \times \mathrm{Pred}_\Lambda$. *That is, for any* $P \in \mathrm{Pred}_\Lambda$ *satisfying*

$$P \sqsubseteq \mathbf{SN} \tag{12}$$

$$P \sqsubseteq S_\delta(P), \tag{13}$$

we have $P \sqsubseteq \mathbf{ON}$. *A predicate that fulfils these conditions is called an* $\mathbf{ON}$*-predicate.*

Proof. By definition, $\mathbf{ON} \sqsubseteq \mathbf{SN}$, hence (12) is fulfilled by $\mathbf{ON}$. For (13), we proceed by cases in the index of $\mathbf{ON}$, i.e., by types.

- Let $t \in \mathbf{ON}_{A_1+A_2}$. Since $t \in \mathbf{SN}$, there is an $i \in \{1,2\}$ and a t' with $t \twoheadrightarrow \kappa_i\, t'$. Moreover, for any such t' we have, as we show now, that $t' \in \mathbf{ON}_{A_i}$. Assume that $t' \notin \mathbf{ON}_{A_i}$ and suppose that $i = 1$ (the case $i = 2$ is symmetric). Then there is a test φ on A_1 with $[\![\varphi]\!] t' \notin \mathbf{SN}$ and, since $[\![[\varphi, \top]]\!]\, t \twoheadrightarrow [\![\varphi]\!]\, t'$, we get a diverging reduction sequence of t under $[\varphi, \top]$. This, however, contradicts that t is observationally normalising, thus t' must be in $\mathbf{ON}_{A_i}$. Putting this together, we have that $t \in \delta^*(S(\mathbf{ON}))_{A_1+A_2}$.

- An analogous argument works for the least fixed point type.

- Let $t \in \mathbf{ON}_{A \to B}$. We need to show that for all $u \in \mathbf{ON}_A$ and $t' : B$ with $t\, u \twoheadrightarrow t'$, $t' \in \mathbf{ON}_B$. Towards a contradiction, assume that there is a test φ on B such that $[\![\varphi]\!] t' \notin \mathbf{SN}$. This implies immediately that $[\![[u]\, \varphi]\!] t \notin \mathbf{SN}$, since $t\, u \twoheadrightarrow t'$, which contradicts that $t \in \mathbf{ON}_{A \to B}$. Therefore, $t' \in \mathbf{ON}_B$.

- For the other coinductive types, the proof works analogously.

Thus we also have $\mathbf{ON} \sqsubseteq S_\delta(\mathbf{ON})$, hence $\Delta(\mathbf{ON}) \sqsubseteq \Psi(\mathbf{ON})$.

We now show that every predicate P with $\Delta(P) \sqsubseteq \Psi(P)$ is included in $\mathbf{ON}$. To this end, let $t \in P_A$ for some type A. We show that $t \in \mathbf{ON}_A$, by showing that for all tests $\varphi : A$, $[\![\varphi]\!]\, t \in \mathbf{SN}$, by induction in φ. Condition (12) gives us that $t \in \mathbf{SN}$, thus $[\![\top]\!]\, t$ and $[\![\bot]\!]\, t$ are strongly normalising. For the induction step, we distinguish the cases for A.

- If $t \in P_{A_1+A_2}$, then there is an $i \in \{1, 2\}$ and a t' with $t \longrightarrow\!\!\!\!\!\rightarrow \kappa_i\, t'$ and $t' \in P_{A_i}$, from condition (13). Hence for any non-trivial test $[\varphi_1, \varphi_2]$, we get by induction that $[\![\varphi_i]\!]\, t' \in \mathbf{SN}$. We show $[\![[\varphi_1, \varphi_2]]\!]\, t \in \mathbf{SN}$ by contradiction. Since $t \in \mathbf{SN}$, there is a normal form $\kappa_i\, r$ with $t \longrightarrow\!\!\!\!\!\rightarrow \kappa_i\, r$. If $[\![[\varphi_1, \varphi_2]]\!]\, t \notin \mathbf{SN}$, then there must be, by confluence, an infinite reduction sequence originating at $[\![[\varphi_1, \varphi_2]]\!]\, (\kappa_i\, r)$. This, however, implies that there is an infinite reduction sequence from $[\![\varphi_i]\!]\, t' \in \mathbf{SN}$, again by confluence. Since this a contradiction, it follows that $[\![[\varphi_1, \varphi_2]]\!]\, t \in \mathbf{SN}$ and $t \in \mathbf{ON}_{A_1+A_2}$.
- The case for least fixed points is treated analogously.
- If $t \in P_{A\to B}$ and $[u]\, \varphi \in \mathrm{Tests}_{A\to B}$, then for all $t' : B$ with $t\, u \longrightarrow\!\!\!\!\!\rightarrow t'$, we have $t' \in P_B$ by (12), in particular, $t\, u \in P_B$ By the induction hypothesis, $[\![\varphi]\!](t\, u) \in \mathbf{SN}$, thus $[\![[u]\, \varphi]\!] t \in \mathbf{SN}$ and $t \in \mathbf{ON}_{A\to B}$.
- The other cases for coinductive types proved analogously.

This shows that $P \sqsubseteq \mathbf{ON}$ and $\mathbf{ON}$ is the largest predicate fulfilling (12) and (13). □

Remark 5.7. In fact, we have shown that $\mathbf{ON}$ is the final (Δ, Ψ)-dialgebra [20]. Alternatively, $\mathbf{ON}$ is the final coalgebra for the functor H on Pred_Λ given by $H(P) = \mathbf{SN} \sqcap S_\delta(P)$, as point-wise intersection is right-adjoint to the diagonal $\Delta : \mathrm{Pred}_\Lambda \to \mathrm{Pred}_\Lambda^2$. We have chosen the characterisation as a coinductive predicate partly in order to avoid introducing unnecessary definitions, but mainly because it facilitates the adoption of up-to techniques to our setting as we show next.

Thm. 5.6 gives us a coinductive proof principle for showing that a term is observationally normalising. However, as with bisimilarity proofs, it can still be quite cumbersome to use this principle. In the remainder of this section, we define a number of up-to techniques for making this task easier. As we are now dealing with predicates, an up-to technique is a functor $T : \mathrm{Pred}_X \to \mathrm{Pred}_X$ on predicates over the family X. We therefore need a slightly different notion of compatibility here. The difference to Sec. 5.2 is that we need another notion of compatibility here.

Definition 5.8. We say that an up-to technique $T \colon \mathrm{Pred}_\Lambda \to \mathrm{Pred}_\Lambda$ is (Δ, Ψ)-*compatible* if $T \circ K_{\mathbf{SN}} \sqsubseteq K_{\mathbf{SN}} \circ T$ and $T \circ S_\delta \sqsubseteq S_\delta \circ T$, that is, $(T \times T) \circ \Psi \sqsubseteq \Psi \circ T$.

The importance of compatible up-to techniques is, just as before, that they are sound and can be composed. The latter is clear, the former is captured in the following lemma.

Lemma 5.9. *If T is (Δ, Ψ)-compatible and $\Delta(P) \sqsubseteq \Psi(T(P))$, then $P \sqsubseteq \mathbf{ON}$.*

Proof. We have that $T \circ \sqcap \sqsubseteq \sqcap \circ (T \times T)$, since $\sqcap$ is a product. Combined with (Δ, Ψ)-compatibility, we find that

$$T \circ \sqcap \circ \Psi \sqsubseteq \sqcap \circ (T \times T) \circ \Psi \sqsubseteq \sqcap \circ \Psi \circ T,$$

hence T is $(\sqcap \circ \Psi)$-compatible for the definition of compatibility from [9].

Using the adjunction $\Delta \dashv \sqcap$, we get from the assumption $\Delta(P) \sqsubseteq \Psi(T(P))$ that $P \sqsubseteq (\sqcap \circ \Psi)(T(P))$, thus, from [9], we find $P \sqsubseteq \mathbf{ON}$. □

The first, very basic, technique we establish is up-to $\mathbf{ON}$. It is given by index-wise union with $\mathbf{ON}$:

$$\mathcal{C}^{\mathbf{ON}}(P) = P \sqcup \mathbf{ON}$$

That $\mathcal{C}^{\mathbf{ON}}$ is an endofunctor on Pred_Λ is immediate. Moreover, we have $\mathcal{C}^{\mathbf{ON}} \circ \mathcal{C}^{\mathbf{ON}} \sqsubseteq \mathcal{C}^{\mathbf{ON}}$ and $\mathrm{Id} \sqsubseteq \mathcal{C}^{\mathbf{ON}}$. Hence $\mathcal{C}^{\mathbf{ON}}$ is a closure operator on predicates over terms. Compatibility is also straightforward to prove.

Proposition 5.10. *$\mathcal{C}^{\mathbf{ON}}$ is (Δ, Ψ)-compatible.*

Proof. Since $\mathbf{ON} \sqsubseteq \mathbf{SN}$, we have $\mathcal{C}^{\mathbf{ON}}(\mathbf{SN}) = \mathbf{SN}$. The second compatibility requirement holds as well:

$$\begin{aligned}
\mathcal{C}^{\mathbf{ON}}(S_\delta(P)) &= S_\delta(P) \sqcup \mathbf{ON} \\
&\sqsubseteq S_\delta(P) \sqcup S_\delta(\mathbf{ON}) && \text{by Thm. 5.6} \\
&\sqsubseteq S_\delta(P \sqcup \mathbf{ON}) && \text{by monotonicity and union} \\
&= S_\delta(\mathcal{C}^{\mathbf{ON}}(P))
\end{aligned}$$

Thus $\mathcal{C}^{\mathbf{ON}}$ is a compatible up-to technique. □

The next technique we consider is the *closure under downwards reductions*. Let $P \sqsubseteq \Lambda$ be a predicate on terms, we define

$$\mathcal{C}^{\downarrow}(P)_A = \{t' : A \mid t \in P_A, t \twoheadrightarrow t'\}.$$

It is easy to see that $\mathcal{C}^{\downarrow}$ is monotone and a closure operator.

Proposition 5.11. *$\mathcal{C}^{\downarrow}$ is (Δ, Ψ)-compatible.*

Proof. The first requirement for (Δ, Ψ)-compatibility says that $\mathcal{C}^{\downarrow}(\mathbf{SN}) \sqsubseteq \mathbf{SN}$. Clearly, if $t \in \mathbf{SN}_A$ and $t \twoheadrightarrow t'$, then $t' \in \mathbf{SN}_A$. Thus the first requirement is fulfilled.

For the second requirement we need to show that $\mathcal{C}^{\downarrow}(S_\delta(P))_A \subseteq S_\delta(\mathcal{C}^{\downarrow}(P))_A$ for all types A. Let $t, t' : A$ with $t \twoheadrightarrow t'$ and $t \in S_\delta(P)_A$. We show that $t' \in S_\delta(\mathcal{C}^{\downarrow}(P))_A$ by case distinction in A.

- $A = B_1 + B_2$. First, we need to find an $i \in \{1, 2\}$ and $s : B_i$ such that $t' \twoheadrightarrow \kappa_i\, s$. Since $t \in S_\delta(P)_A$, there exist i and s' with $t \twoheadrightarrow \kappa_i\, s'$. By confluence, there is an s with $\kappa_i\, s' \twoheadrightarrow \kappa_i\, s \twoheadleftarrow t'$, giving us the required term $s : B_i$.

  Secondly, we need to show that for all $s : B_i$ with $t' \twoheadrightarrow \kappa_i\, s$, we have $s \in \mathcal{C}^{\downarrow}(P)_{B_i}$. So let $s : B_i$ with $t' \twoheadrightarrow \kappa_i\, s$. Since $t \twoheadrightarrow t'$, we have $t \twoheadrightarrow t' \twoheadrightarrow \kappa_i\, s$. Therefore, $s \in P_{B_i}$ because $t \in S_\delta(P)_A$. Using that $\mathcal{C}^{\downarrow}$ is a closure operator, we find $s \in \mathcal{C}^{\downarrow}(P)_{B_i}$.

  Putting these two arguments together, $t' \in S_\delta(P)_{B_1 + B_2}$.

- The case for least fixed points is analogous.

- $A = B \to C$. Here, we need to show that for all $u \in \mathbf{ON}_B$, all terms $s : C$ with $t'\, u \twoheadrightarrow s$ are in $\mathcal{C}^{\downarrow}(P)_C$, that is, $s \in \mathcal{C}^{\downarrow}(P)_C$. Since we have for such s that $t\, u \twoheadrightarrow t'\, u \twoheadrightarrow s$, s must be in P_C already and the claim follows.

- The other coinductive types follow analogously.

This proves that $\mathcal{C}^{\downarrow} \circ S_\delta \sqsubseteq S_\delta \circ \mathcal{C}^{\downarrow}$, thus $\mathcal{C}^{\downarrow}$ is (Δ, Ψ)-compatible. □

The closure under reductions already lifts quite a burden from us in showing that a term is observationally normalising. However, very often we would like to construct a predicate P that contains, for example, a term $t \in P_{A_1 \times A_2}$ and terms $r_i \in P_{A_i}$ with $\pi_i\, t \succ r_i$. The problem is that we then also need to have $\pi_i\, t \in P_{A_i}$ and then also $e[\pi_i\, t] \in P_B$ for evaluation contexts e. Explicitly adding all these terms can be avoided due to the following up-to technique.

Let $\mathcal{C}^{\mathrm{ev}}$ be the *closure under evaluations* given by

$$\begin{aligned}
\mathcal{C}^{\mathrm{ev}}(P)_A = & \bigcup_{i=1,2} \{ \pi_i\, t \mid t \in P_{A_1 \times A_2} \wedge A = A_i \wedge \exists t' \in P_{A_i}.\, \pi_i\, t \succ t' \} && (14) \\
& \cup \{ \xi\, t \mid t \in P_{\nu X.B} \wedge A = B[\nu X.B/X] \wedge \exists t' \in P_A.\ \xi\, t \succ t' \} && (15) \\
& \cup P_A. && (16)
\end{aligned}$$

That $\mathcal{C}^{\mathrm{ev}}$ is a functor $\mathrm{Pred}_\Lambda \to \mathrm{Pred}_\Lambda$ can be seen rather easily and, moreover, we have $\mathrm{Id} \sqsubseteq \mathcal{C}^{\mathrm{ev}}$ by definition. We now show that $\mathcal{C}^{\mathrm{ev}}$ is a compatible up-to technique.

Proposition 5.12. *$\mathcal{C}^{\mathrm{ev}}$ is (Δ, Ψ)-compatible.*

Proof. First, we show $\mathcal{C}^{\mathrm{ev}}(\mathbf{SN})_A \subseteq \mathbf{SN}_A$ by the cases of the definition of $\mathcal{C}^{\mathrm{ev}}$.

(14) Let $\pi_i\, t \in \mathcal{C}^{\mathrm{ev}}(\mathbf{SN})_{A_i}$ where $t \in \mathbf{SN}_{A_1 \times A_2}$ and $\pi_i\, t \succ t'$ with $t' \in \mathbf{SN}_{A_i}$. We note that either $t = e[f]$ with $f \in \mathrm{Sig}$ or $t = e[\lambda D]$ for some (possibly empty) evaluation context e, by definition of contraction. In both cases it is clear, from $t' \in \mathbf{SN}$ and $t \in \mathbf{SN}$, that $\pi_i\, t \in \mathbf{SN}$, since the only reductions possible on this term are either reduction on t or factor through the contraction to t'.

(15) Let $\xi\, t \in \mathcal{C}^{\mathrm{ev}}(\mathbf{SN})$ where $t \in \mathbf{SN}_{\nu X.B}$ and $\xi\, t \succ t'$ with $t' \in \mathbf{SN}_{B[\nu X.B/X]}$. Then $\xi\, t \in \mathbf{SN}_{\nu X.B}$ by an analogous argument.

(16) In the last case, $t \in \mathbf{SN}_A$ holds by definition.

The second part is to show that $\mathcal{C}^{\mathrm{ev}}(S_\delta(P))_A \subseteq S_\delta(\mathcal{C}^{\mathrm{ev}}(P))_A$, which we again do by distinguishing the cases in the definition of $\mathcal{C}^{\mathrm{ev}}$.

(14) Let $\pi_i\, t \in \mathcal{C}^{\mathrm{ev}}(S_\delta(P))_{A_i}$ where $t \in S_\delta(P)_{A_1 \times A_2}$ and $\pi_i\, t \succ t'$ with $t' \in S_\delta(P)_{A_i}$. To show $\pi_i\, t \in S_\delta(\mathcal{C}^{\mathrm{ev}}(P))_{A_i}$, we distinguish the cases for A_i.

- $A_i = B_1 + B_2$. We have to show that $\pi_i\, t$ has WHNF with the constructor argument being in P_{B_k} for some k and that any such argument is in P_{B_k}.

  For the first part, we note that, since $t' \in S_\delta(P)_{B_1 + B_2}$, there is an $s \in P_{B_k}$ with $t' \twoheadrightarrow \kappa_k\, s$. Therefore, $\pi_i\, t \succ t' \twoheadrightarrow \kappa_k\, s$, giving us the required WHNF for $\pi_i\, t$.

  Next, let $r : B_k$ with $\pi_i\, t \twoheadrightarrow \kappa_k\, r$. There is a term s, as in the following diagram, through which this reduction sequence factors.

$$\begin{tikzcd}
\pi_i\, t \arrow[r, two heads] \arrow[d, phantom, sloped, "\succ"] & \pi_i\, t'' \arrow[d, phantom, sloped, "\succ"] & \\
t' \arrow[r, two heads] & s \arrow[r, two heads] & \kappa_k\, r
\end{tikzcd}$$

It is given as follows. If $t = e\,[\lambda\{q_1 \mapsto u_1 \;;\; \cdots \;;\; q_n \mapsto u_n\}]$, and $t' = u_j[\sigma]$ such that $q_j[\sigma] = \pi_i\, e$, then $t'' = e'\,[\lambda\{q_1 \mapsto u''_1 \;;\; \cdots \;;\; q_n \mapsto u''_n\}]$, and $s = u''_j[\sigma']$, where $\sigma \twoheadrightarrow \sigma'$ is given by Lem. 2.9. If $t = e[f]$, the an analogous argument holds for $t'' = e'[f]$. Using said term s, we get a reduction sequence $t' \twoheadrightarrow \kappa_k\, r$, hence $r \in P_{B_k}$, as $t' \in P_{B_1+B_2}$.

Thus $\pi_i\, t$ is already in $S_\delta(P)_{B_1+B_2}$, hence in $S_\delta(\mathcal{C}^{\mathrm{ev}}(P))_{A_i}$.

- $A_i = B \to C$. Let $u \in \mathbf{ON}_B$ and $(\pi_i\, t)\, u \twoheadrightarrow r$. Similar to the sum case, we factor this reduction sequence as

$$\begin{tikzcd}
(\pi_i\, t)\, u \arrow[r, two heads] \arrow[d, phantom, "\curlyvee"] & (\pi_i\, t'')\, u \arrow[d, phantom, "\curlyvee"] & \\
t'\, u \arrow[r, two heads] & s\, u \arrow[r, two heads] & r.
\end{tikzcd}$$

It follows that $r \in P_C \subseteq \mathcal{C}^{\mathrm{ev}}(P)_C$. Since this holds for any $u \in \mathbf{ON}_B$, we have $\pi_i\, t \in S_\delta(\mathcal{C}^{\mathrm{ev}}(P))_{B\to C}$.

- The other cases for A_i are treated analogously.

Thus, if $\pi_i\, t \in \mathcal{C}^{\mathrm{ev}}(S_\delta(P))_{A_i}$ by (14), then $\pi_i\, t \in S_\delta(\mathcal{C}^{\mathrm{ev}}(P))_{A_i}$.

(15) The proof in this case is analogous to that for the case (14).

(16) This case is trivial by definition.

Hence, in all cases for $\mathcal{C}^{\mathrm{ev}}$, we find $\mathcal{C}^{\mathrm{ev}}(S_\delta(P))_A \subseteq S_\delta(\mathcal{C}^{\mathrm{ev}}(P))_A$, thus $\mathcal{C}^{\mathrm{ev}} \circ S_\delta \sqsubseteq S_\delta \circ \mathcal{C}^{\mathrm{ev}}$. Combined with the first part, we have that $\mathcal{C}^{\mathrm{ev}}$ is (δ, Ψ)-compatible. □

Since $\mathcal{C}^{\mathbf{ON}}$, $\mathcal{C}^{\mathrm{ev}}$ and $\mathcal{C}^{\downarrow}$ are compatible, we can combine them into a single up-to technique $\mathcal{C} = \mathcal{C}^{\downarrow} \circ \mathcal{C}^{\mathrm{ev}} \circ \mathcal{C}^{\mathbf{ON}}$. We illustrate the proof principle for observational normalisation together with this combined up-to technique in the following example.

Example 5.13. This example consists of three parts. We first show how this general up-to technique can be used to establish observational normalisation of a definition in the *simple stream SOS format* [29]. Then we extract an up-to technique from this definition. Finally, we use this technique to prove another term observationally normalising, even though its definition is not in the simple SOS format.

1. We begin by defining negation on Bool and its point-wise lifting to streams.

$$\begin{aligned}
&\neg : \mathrm{Bool} \to \mathrm{Bool} && \sim\ : \mathrm{Bool}^\omega \to \mathrm{Bool}^\omega \\
&\neg\top = \bot && \mathrm{hd}(\sim s) = \neg(\mathrm{hd}\, s) \\
&\neg\bot = \top && \mathrm{tl}(\sim s) = \sim(\mathrm{tl}\, s)
\end{aligned}$$

It is clear that $\neg t \in \mathbf{SN}_{\mathrm{Bool}}$ for every strongly normalising t, hence $\neg \in \mathbf{ON}$. Using this, we can also show that $\sim$ is observationally normalising.

Proof. It suffices to show that the predicate P given by

$$\begin{aligned}
P_{\mathrm{Bool}^\omega \to \mathrm{Bool}^\omega} &= \{\sim\} \\
P_{\mathrm{Bool}^\omega} &= \{\sim s \mid s \in \mathbf{ON}_{\mathrm{Bool}^\omega}\} \\
P_A &= \emptyset, && \text{all other } A
\end{aligned}$$

is an $\mathbf{ON}$-predicate up to $\mathcal{C}$. Indeed, we have $P \sqsubseteq \mathbf{SN}$ by definition. Moreover, for all $s \in \mathbf{ON}_{\mathrm{Bool}^\omega}$ and $t : \mathrm{Bool}^\omega$ with $\sim s \twoheadrightarrow t$, we have $t \in \mathcal{C}^{\downarrow}(P)_{\mathrm{Bool}^\omega}$, hence $t \in \mathcal{C}(P)_{\mathrm{Bool}^\omega}$. To show that P is also closed from Bool^ω on, we note that for all $s \in \mathbf{ON}_{\mathrm{Bool}^\omega}$, $\mathrm{hd}(\sim s) \succ \neg(\mathrm{hd}\, s)$ and $\mathrm{tl}(\sim s) \succ \sim(\mathrm{tl}\, s)$. Since $\neg(\mathrm{hd}\, s) \in \mathbf{ON}_{\mathrm{Bool}}$, by the discussion above, we have $\mathrm{hd}(\sim s) \in (\mathcal{C}^{\mathrm{ev}} \circ \mathcal{C}^{\mathbf{ON}})(P)_{\mathrm{Bool}}$, hence for all $t : \mathrm{Bool}$ with $\mathrm{hd}(\sim s) \twoheadrightarrow t$, we have $t \in \mathcal{C}(P)_{\mathrm{Bool}}$. Likewise, we have $\sim(\mathrm{tl}\, s) \in P_{\mathrm{Bool}^\omega}$, since $\mathrm{tl}\, s \in \mathbf{ON}_{\mathrm{Bool}^\omega}$, thus $t \in \mathcal{C}(P)_{\mathrm{Bool}^\omega}$ for all t with $\mathrm{tl}(\sim s) \twoheadrightarrow t$. Summarising, we have $\Delta(P) \sqsubseteq \Psi(\mathcal{C}(P))$, hence $\sim\, \in \mathbf{ON}$. □

2. The next step is to derive an up-to technique from the definition of $\sim$, which we can use to prove observational normalisation of other terms. The technique is given by $\mathcal{C}^{\sim} = \mathcal{C}^{\downarrow} \circ T$, where

$$\begin{aligned} T(P)_{\mathrm{Bool}^\omega} &= \{\mathrm{tl}^n(\sim s) \mid n \in \mathbb{N}, s : \mathrm{Bool}^\omega, \mathrm{tl}^n\, s \in P_{\mathrm{Bool}^\omega}\} \\ T(P)_{\mathrm{Bool}} &= \{\mathrm{hd}(\mathrm{tl}^n(\sim s)) \mid n \in \mathbb{N}, s : \mathrm{Bool}^\omega, \mathrm{hd}(\mathrm{tl}^n\, s) \in P_{\mathrm{Bool}^\omega}\}. \end{aligned}$$

We claim that $\mathcal{C}^{\sim}$ is (Δ, Ψ)-compatible.

Proof. The first condition, $\mathcal{C}^{\sim}(\mathbf{SN}) \sqsubseteq \mathbf{SN}$, is clear by definition. To show the second condition, note that if $\mathrm{tl}^n\, s \in S_\delta(P)_{\mathrm{Bool}^\omega}$, then for all t with $\mathrm{tl}(\mathrm{tl}^n\, s) \twoheadrightarrow t$, we have $t \in P_{\mathrm{Bool}^\omega}$. In particular, $\mathrm{tl}^{n+1}\, s \in P_{\mathrm{Bool}^\omega}$, which implies that $\mathrm{tl}^{n+1}(\sim s) \in T(P)_{\mathrm{Bool}^\omega}$. From here, it follows, by the definition of $\mathcal{C}^{\sim}$ as composition of T with the reduction closure, that any r with $\mathrm{tl}(\mathrm{tl}^n(\sim s)) \twoheadrightarrow r$ is in $S_\delta(\mathcal{C}^{\sim}(P))_{\mathrm{Bool}^\omega}$. Analogously, every r' with $\mathrm{hd}(\mathrm{tl}^n(\sim s)) \twoheadrightarrow r'$ is also in $S_\delta(\mathcal{C}^{\sim}(P))_{\mathrm{Bool}}$. Thus $\mathcal{C}^{\sim}$ is compatible. □

3. Finally, we use the up-to technique $\mathcal{C}^{\sim}$, combined with the general up-to techniques from above, to show that the following definition of the alternating bit stream is observationally normalising.

$$\mathrm{alt} : \mathrm{Bool}^\omega \qquad\qquad \mathrm{hd}\,\mathrm{alt} = \top \qquad\qquad \mathrm{tl}\,\mathrm{alt} = \sim \mathrm{alt}$$

Note that the definition of alt is not in the simple stream SOS format because tl alt is defined as a term of depth 2 (not 1). It is in the slightly more general format where the tail (or derivative) can be defined by any term over the signature which in this case contains the symbols $\sim$ and alt*. The proof that alt is observationally normalising is now as simple as it could be: The predicate P, where $P_{\mathrm{Bool}^\omega} = \{\mathrm{alt}\}$ and $P_A = \emptyset$ for all other types A, is an $\mathbf{ON}$-predicate up to $\mathcal{C} \circ \mathcal{C}^{\sim}$.

Proof. First, we have $\mathrm{alt} \in \mathbf{SN}$ by its definition. Second, we find that $\top \in \mathbf{ON}_{\mathrm{Bool}}$, hence $\mathrm{hd}\,\mathrm{alt} \in \mathcal{C}^{\mathrm{ev}}(\mathcal{C}^{\mathbf{ON}}(P))_{\mathrm{Bool}}$. Moreover, since $\mathrm{alt} \in P_{\mathrm{Bool}^\omega}$, we have that $\sim\mathrm{alt} = \mathrm{tl}^0(\sim\mathrm{alt})$ is in $\mathcal{C}^{\sim}(P)_{\mathrm{Bool}^\omega}$, thus $\mathrm{tl}\,\mathrm{alt} \in \mathcal{C}^{\mathrm{ev}}(\mathcal{C}^{\sim}(P))_{\mathrm{Bool}^\omega}$. Putting this together, we have $P \sqsubseteq S_\delta(\mathcal{C}(\mathcal{C}^{\sim}(P)))$, hence $\mathrm{alt} \in \mathbf{ON}_{\mathrm{Bool}^\omega}$. □

Let us make some remarks about Ex. 5.13 before we close this section. Note that we have derived, in the second step, an up-to technique from the definitions given in the first step, and used this technique in the last step to close the predicate P so that terms in P

* This generalisation is an analogue of the generalisation from the simple SOS format that corresponds to a natural transformation $\Sigma(\mathrm{Bool} \times (-)) \Longrightarrow \mathrm{Bool} \times \Sigma(-)$ to the format $\Sigma(\mathrm{Bool} \times (-)) \Longrightarrow \mathrm{Bool} \times \Sigma^*(-)$ where Σ^* is the free monad over Σ.

can occur in the context of $\sim$. In the light of Ex. 3.2, it is clear that we cannot obtain up-to techniques by just taking the contextual closure of a predicate, so that [9, Sec. 4.3] unfortunately does not apply immediately. However, we expect to be able to derive up-to techniques, which are similar to contextual closures, for more general declarations, only that we need to impose extra conditions on the terms we take the closure on. For instance, we can only safely put, for a term t, $\mathrm{even}\, t$ into the closure, if $\mathrm{tl}\, t$ is fully defined. Finally, note that a particularly nice feature of compatible up-to techniques is that they can be composed. This allows us to derive up-to techniques separately for each declaration and then compose them, to form an up-to technique for large declaration blocks.

6 (Un)Decidability of Observational Equivalence

In this section, we present two results concerning decidability of observational equivalence. The first just makes the intuitive assertion precise that, in general, observational equivalence is undecidable. The second result, however, establishes a fragment of the language on which observational equivalence actually becomes decidable. This fragment is admittedly rather small, but still an interesting start.

6.1 Observational Equivalence is Undecidable

Proposition 6.1. *Observational equivalence is semi-decidable.*

Proof. Let t_1, t_2 be two terms of type A. To decide whether $t_1 \not\equiv_{\mathrm{obs}} t_2$ we can enumerate all tests on A and check for each of them whether t_1 and t_2 do not satisfy it simultaneously. This gives a procedure that terminates, if $t_1 \not\equiv_{\mathrm{obs}} t_2$. We now show that it is impossible to give a general algorithm that checks observational inequivalence via an encoding of Post's correspondence problem (PCP). This shows that observational equivalence is undecidable.

Let A be a finite alphabet with at least two letters, and $w = (w_1, \dots, w_N)$ and $v = (v_1, \dots, v_N)$ sequences of words over A. A solution to the PCP for (w, v) is a finite sequence $(i_k)_{1 \leq k \leq K}$ such that for all $k \in \{1, \dots, K\}$, $1 \leq i_k \leq N$, and

$$w_{i_1} \cdots w_{i_K} = v_{i_1} \cdots v_{i_K}. \tag{17}$$

It is known to be undecidable whether a solution exists for any given (w, v). The idea of the encoding of the PCP, which we are about to give, is to define a decidable predicate on finite sequences that contains all lists for which (17) holds. This is carried out in the following way.

We begin by defining the relevant data structures as types, and basic functions on them. Let A have n letters, so that we can encode A as the sum $A = \underbrace{1 + \cdots + 1}_{n}$. Words over A are given by lists $A^* = \mu X.1 + A \times X$, thus a word w can be written as a sequence of constructors, and concatenation of lists $\cdot$ can be defined inductively in the usual way. We can also define predicates $eq_A : A^2 \to \mathrm{Bool}$ and $eq_L : (A^*)^2 \to \mathrm{Bool}$ that are computable on observationally normalising arguments, such that

$$\begin{aligned} eq_A(a, b) \equiv \top \quad &\text{iff} \quad a \equiv b \qquad \text{and} \\ eq_L(u, v) \equiv \top \quad &\text{iff} \quad u \equiv v. \end{aligned}$$

Finally, we encode numbers bounded by N as $\underline{N} = \underbrace{1 + \cdots + 1}_{N}$ and finite, non-empty sequences of them by $\underline{N}^+ = \mu X.(\underline{N} + \underline{N} \times X)$.

To reduce an instance (w, v) of PCP to observational equivalence, we define a map h which given a sequence $u = (i_k)_{1 \leq k \leq K}$ computes the pair $(w_{i_1} \cdots w_{i_K}, v_{i_1} \cdots v_{i_K})$, a predicate P which tests whether a sequence is a solution, and the empty predicate $P_\bot$ by

$$\begin{aligned} & h : \underline{N}^+ \to A^* \times A^* \\ & h(i) = (w_i, v_i) \\ & h(i : u) = (w_i \cdot (\pi_1\,(h\,u)), v_i \cdot (\pi_2\,(h\,u))) \end{aligned}$$

$$\begin{aligned} & P : \underline{N}^+ \to \mathrm{Bool} \qquad\qquad && P_\bot : \underline{N}^+ \to \mathrm{Bool} \\ & P = eq_L \circ h && P_\bot(u) = \bot \end{aligned}$$

Hence, for $u : \underline{N}^+$, $P(u) \equiv \top$ if and only if u solves the PCP for (w, v), and P is clearly computable for **ON** terms. Moreover, since observational inequivalence is witnessed by tests, $P \not\equiv_{\mathrm{obs}} P_\bot$ means that there is a $\varphi \in \mathrm{Tests}_{\underline{N}^+ \to \mathrm{Bool}}$ such that $P \vDash \varphi$ and $P_\bot \not\vDash \varphi$. We may assume that such a φ is of the form $\varphi = [u]\,[\top, \bot]$ for some $u : \underline{N}^+$ so that $P(u) \equiv \top$, and u is a solution to the PCP. Thus, if we can find φ, we can solve the PCP. In other words, $P \not\equiv_{\mathrm{obs}} P_\bot$ if and only if the PCP for (w, v) has a solution.

In summary, if for any terms t_1, t_2 it is decidable whether there is a test that distinguishes t_1 and t_2, then the PCP is decidable, hence observational equivalence cannot be decidable. □

6.2 Decidability on a Language Fragment

Even though observational equivalence is undecidable on the full language, there is a (tiny) fragment of the language on which we can decide it. Analysing the encoding of Post's correspondence problem, we find that the encoding crucially requires functions. Indeed, once we forbid terms of function type, observational equivalence becomes decidable.

Therefore, we assume, for this section, that all terms do not use abstraction and application for functions. Moreover, we again restrict to observationally normalising terms. The first assumption has the implication that abstractions (λ and definition blocks) only use copatterns for products and greatest fixed points but never patterns, thus the terms cannot contain variables.

We make three more simplifying assumptions: First, we fix a definition block Σ, and assume that all terms t are typed within Σ and have no further **rlet**-bindings. Second, all copatterns shall only have one layer, that is, bodies are always of the form $\{\pi_1 \cdot \mapsto t_1 \ ; \ \pi_2 \cdot \mapsto t_2\}$ or $\{\xi \cdot \mapsto t\}$. Third, we assume that the λ-abstraction is never used. These three assumptions do not pose any limitations, as we can first unroll nested copatterns into nested λ-abstractions and then introduce for each λ-abstraction a new symbol into the signature. These transformations preserve observational equivalence.

More precisely, we assume that terms are of the form

$$t ::= f \mid \kappa_i\, t \mid \alpha\, t \mid \pi_i\, t \mid \xi\, t$$

and are typed within a fixed definition block Σ, in which all definitions are of the form

$$D ::= \{\pi_1 \cdot \mapsto t_1 \ ; \ \pi_2 \cdot \mapsto t_2\} \mid \{\xi \cdot \mapsto t\}.$$

Under these assumptions, Algorithm 1 decides whether two terms are observationally equivalent, returning a witnessing test if they are not or a bisimulation up-to convertibility if they are. The notation we use in the algorithm is similar to that of monadic Haskell-code, where we treat Tests + (−) as a monad and use the left-arrow notation.*

Algorithm 1: Decide whether two terms are observationally equivalent

```
CheckBisim($t_1, t_2 \in \mathbf{ON}_A$, $R \in \mathrm{Rel}(\Lambda)$) : $\mathrm{Tests} + \mathrm{Rel}(\Lambda)$
    Invariant: If $t_1 \; R_A \; t_2$, then $R$ is a bisimulation up-to convertibility.
    if $t_1 \; R_A \; t_2$ then return $R$
    Add $(t_1, t_2)$ to $R$
    Bring $t_1$ and $t_2$ into WHNF
    case $t_i = f_i$ with $(f_i : A = D_i) \in \Sigma$
        case $D_1 = \{\pi_1 \mapsto r_1 \; ; \; \pi_2 \mapsto r_2\}$ and $D_2 = \{\pi_1 \mapsto s_1 \; ; \; \pi_2 \mapsto s_2\}$
            $R' \leftarrow$ UpdateTest($\boldsymbol{\lambda}\varphi. [\pi_1] \, \varphi$, CheckBisim($r_1$, $s_1$, $R$))
            UpdateTest($\boldsymbol{\lambda}\varphi. [\pi_2] \, \varphi$, CheckBisim($r_2$, $s_2$, $R'$))
        case $D_1 = \{\xi \mapsto r\}$ and $D_2 = \{\xi \mapsto s\}$
            UpdateTest($\boldsymbol{\lambda}\varphi. [\xi] \, \varphi$, CheckBisim($r$, $s$, $R$))
    case $t_1 = \kappa_i \, t_1'$ and $t_2 = \kappa_j \, t_2'$
        if $i \neq j$ then return $[\top, \bot]$
        UpdateTest(CoprodTest($i$), CheckBisim($t_1'$, $t_2'$, $R$))
    case $t_1 = \alpha \; t_1'$ and $t_2 = \alpha \; t_2'$
        UpdateTest($\boldsymbol{\lambda}\varphi. \alpha^{-1} \, \varphi$, CheckBisim($t_1'$, $t_2'$, $R$))
end
UpdateTest($f$ : Tests $\to$ Tests, $U \in \mathrm{Tests} + \mathrm{Rel}(\Lambda)$) : $\mathrm{Tests} + \mathrm{Rel}(\Lambda)$
    case $U$ is a test $\varphi$ return $f(\varphi)$
    case $U$ is a relation $R$ return $R$
end
CoprodTest($i$, $\varphi$)
    if $i = 1$ then return $[\varphi, \bot]$ else return $[\bot, \varphi]$
end
```

Informally, the algorithm works as follows. It compares recursively the given terms according to what it requires to fulfil the same tests. If that fails, it builds up a test witnessing this, while returning from the recursion. Otherwise, it puts the given pair of terms in the bisimulation candidate R and tries to close R recursively. Once it arrives at a pair that has already been compared, it returns the constructed relation, as it is closed at that point.

Termination of the algorithm is ensured by the fact that, as we do not allow the use of functions, a term t in the fixed definition block Σ is essentially a finite transition system. Modulo reduction to WHNF, the only way of creating a term of inductive type is by a finite sequence of constructors, hence we can remove only finitely many such. For coinductive types, on the other hand, a WHNF must be a symbol in Σ, hence we must eventually reach a pair of symbols that are already in the relation, as there are only $|\Sigma|^2$ such pairs. Therefore, the algorithm terminates.

We will now make these arguments precise.

Theorem 6.2. *Let $t_1, t_2 : A$ be any two observationally normalising terms in the restricted language. Then the following holds.*

* Implementation: `https://github.com/hbasold/Sandbox/blob/master/OTTTests/DecideEquiv.hs`

(i) If `CheckBisim`$(t_1, t_2, \emptyset)$ *returns a test* φ*, then* $t_1 \models \varphi \neq t_2 \models \varphi$*.*

(ii) If `CheckBisim`$(t_1, t_2, \emptyset)$ *returns a relation* R*, then* R *is a bisimulation up-to convertibility and* $(t_1, t_2) \in R_A$*.*

(iii) `CheckBisim`$(t_1, t_2, \emptyset)$ *terminates.*

Proof. (i) This is very easy to see, as we only stop with a test if the constructors for elements of a sum type do not match and then trace back the observations we made to get to the sum constructors.

(ii) We prove the invariant given at the beginning of `CheckBisim`: If t_1 and t_2 are related by R, then R is already a bisimulation up-to convertibility. Since $\emptyset$ fulfils this and we return R without further changes, the statement of the theorem follows.

So assume that t_1 and t_2 are not yet related by R, in which case the pair is added and we continue on the WHNF of these terms. In all cases, we recurse on elements of $\delta(t_1)$ and $\delta(t_2)$. This means that, in the recursion step, if these elements are already in R, we indeed have found a bisimulation. For example, if $A = B_1 \times B_2$ and $t_i = f_i$ with $(f_i : A = D_i) \in \Sigma$, then $\delta(t_i)(j) = \{t' : B_j \mid t' \equiv \pi_j\, t\}$ for $j = 1, 2$ and, in particular, $r_j \in \delta(f_1)(j)$ and $s_j \in \delta(f_2)(j)$. Since, as a result of `CheckBisim`(r_1, s_1, R), R' is a bisimulation up-to convertibility and contains (r_1, s_1), the result of `CheckBisim`(r_2, s_2, R') contains (r_1, s_1), (r_2, s_2) and is a bisimulation up-to convertibility as well. Therefore, the invariant is preserved.

(iii) We use the following two termination measures: n, the maximum of the sizes of the terms t_1 and t_2, and $m = |\Sigma|^2 - \#$pairs of symbols in R. On the recursive calls of `CheckBisim` for inductive types, n strictly decreases and for coinductive types, m strictly decreases (though n might increase in this case). Thus m becomes eventually 0, meaning that all symbols of Σ have been related with each other. From here on, n must decrease until it becomes 1, at which point t_1 and t_2 must be symbols from Σ and are thus related. Hence, `CheckBisim` stops and returns R. □

After having proved that we can decide observational equivalence on observationally normalising terms, one might ask whether observational normalisation is a decidable property. The answer to this question is indeed yes and we describe the idea for a decision procedure in the following.

We have seen that **ON** is the largest predicate that is contained in **SN** and is closed under δ-steps. This can be leveraged, just as we did for observational equivalence, by constructing recursively a predicate that contains strongly normalising terms, giving, again as before, a terminating procedure, if we can decide strong normalisation. In the restricted calculus, we only only need to decide whether there is a WHNF though, since

1. there is always a unique reduction sequence and
2. we check strong normalisation by continuing to check recursively for observational normalisation under constructors.

So we are left with the task to decide whether a term has a WHNF. This can be done by trying to reduce the term to a WHNF and storing every term in the reduction sequence in a predicate that shall witness whether a term has *no* WNHF. If we reach a term a second time in the reduction sequence, we know that there can be no WHNF. For the same reasons as before, this eventually terminates due to the finite automaton structure of the terms.

We illustrate this with an example of a term that has no WHNF.

Example 6.3. Let Σ be the following declaration block.

$$\begin{aligned}
&\mathrm{grow} : \mathbb{N} \times \mathbb{N}^{\omega} \\
&\pi_1\,\mathrm{grow} = 0 \\
&\pi_2\,\mathrm{grow} = \pi_2\,(\xi\,(\pi_2\,\mathrm{grow}))
\end{aligned}$$

The term $\pi_2\,\mathrm{grow}$ leads to the following reduction sequence

$$\pi_2\,\mathrm{grow} \longrightarrow \pi_2\,(\xi\,(\pi_2\,\mathrm{grow})) \longrightarrow \pi_2\,(\xi\,(\pi_2\,(\xi\,(\pi_2\,\mathrm{grow})))) \longrightarrow \cdots$$

that is obviously diverging. We can show this with the following predicate, which is constructed by the decision procedure.

$$\begin{aligned}
P_{\mathbb{N}^{\omega}} &= \{\pi_2\,\mathrm{grow}, \pi_2\,(\xi\,(\pi_2\,\mathrm{grow}))\} \\
P_{\mathbb{N}\times\mathbb{N}^{\omega}} &= \{\xi\,(\pi_2\,\mathrm{grow})\} \\
P_C &= \emptyset, \text{ all other types } C
\end{aligned}$$

That P indeed proves that $\pi_2\,\mathrm{grow}$ has no WHNF is seen as follows. In order to reduce $\pi_2\,\mathrm{grow}$, we need to reduce $\pi_2\,(\xi\,(\pi_2\,\mathrm{grow}))$, thus we need to make a reduction step on $\xi\,(\pi_2\,\mathrm{grow})$, which in turn needs a reduction of $\pi_2\,\mathrm{grow}$. Since all of these terms are in P, we have found a loop in the reduction sequence, hence $\pi_2\,\mathrm{grow}$ has WHNF. □

7 Conclusion and Future Work

We have introduced the notions of observational normalisation and observational equivalence for mixed inductive-coinductive programs, and argued for their suitability. Roughly, we have shown that observational equivalence is sound and complete for equality in Cartesian closed μ-bicomplete categories [13], a statement we would like to make precise in the future. Furthermore, we have given proof methods for observational equivalence and observational normalisation that are based on coalgebraic techniques.

We believe that this semantic approach can used to extend our results to other languages and different notions of program equivalences. We also believe that our approach using tests is suitable for implementation in theorem provers. To support this claim, we have implemented tests and observational equivalence in Agda.* We would like to investigate the possibility of implementing the proof principles and up-to techniques from Section 5 as proof strategies in theorem provers. This can be achieved by defining a syntax for **ON**-predicates, bisimulations and the corresponding up-to techniques. Then a declaration could be registered as being well-formed by exhibiting an **ON**-predicate (up to some technique) in this syntax. Similar in spirit to [6], syntactic proofs of observational equivalence could be used to coerce terms of dependent type.

Closely related to this is the need to extend our work to richer type systems, for example, with dependent types. The conditions to carry out such an extension are that the reduction relation of the calculus in question is confluent, and that we can find WHNFs for inductive types and can apply destructors to coinductive types. Even more generally,

* `https://github.com/hbasold/Sandbox/blob/master/OTTTests/Tests.agda`

it might be possible to derive the test logic from a coalgebra, like the one we defined. The only problem is that we cannot define observational normalisation this way, as it is part of the type of the term coalgebra. It needs to be investigated if there is an abstract characterisation of observations that allows us to also define observational normalisation.

Another direction for extending our results would be to relax the definition of observational normalisation so that we only require an **ON**-predicate P to be a subset of weak head normalising terms in Thm. 5.6. This would simplify proofs of observational normalisation significantly, as we do not need to prove strong normalisation separately. Note that this is closely related to proofs of strong normalisation by using reducibility candidates [18].

References

[1] A. Abel. Type-based termination, inflationary fixed-points, and mixed inductive-coinductive types. *Electron. Proc. Theor. Comput. Sci.*, 77:1–11, Feb. 2012.

[2] A. Abel and B. Pientka. Wellfounded recursion with copatterns: a unified approach to termination and productivity. In *ICFP*, pages 185–196, 2013.

[3] A. Abel, B. Pientka, D. Thibodeau, and A. Setzer. Copatterns: Programming infinite structures by observations. In *Proc. of POPL*, pages 27–38. ACM, 2013.

[4] S. Abramsky. The lazy lambda calculus. In *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.

[5] J. Adámek, S. Milius, and J. Velebil. Semantics of higher-order recursion schemes. *Log. Methods Comput. Sci.*, 7(1), 2011.

[6] T. Altenkirch, C. McBride, and W. Swierstra. Observational equality, now! In *Proc. of PLPV '07*, pages 57–68. ACM, 2007.

[7] H. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types.* Cambridge University Press, Cambridge ; New York, July 2013.

[8] H. P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* North Holland, Amsterdam; New York; Oxford, revised edition edition, Nov. 1985.

[9] F. Bonchi, D. Petrisan, D. Pous, and J. Rot. Coinduction up to in a fibrational setting. *ArXiv14016675 Cs*, pages 1–9, 2014.

[10] H. Cirstea and G. Faure. Confluence of pattern-based calculi. In F. Baader, editor, *Term Rewriting and Applications*, number 4533 in Lecture Notes in Computer Science, pages 78–92. Springer Berlin Heidelberg, Jan. 2007.

[11] P.-L. Curien and R. D. Cosmo. A confluent reduction for the λ-calculus with surjective pairing and terminal object. *J. Funct. Program.*, 6(02):299–327, 1996.

[12] M. Escardó. Synthetic topology: of data types and classical spaces. *ENTCS*, 87:21–156, Nov. 2004.

[13] J. Fortier and L. Santocanale. Cuts for circular proofs: semantics and cut-elimination. In *CSL*, pages 248–262, 2013.

[14] H. Geuvers. Inductive and coinductive types with iteration and recursion. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs, Bastad*, pages 193–217, 1992.

[15] N. Ghani. Beta-eta equality for coproducts. In *Proceedings of TLCA'95*, Lecture Notes in Computer Science, pages 171–185. Springer-Verlag, 1995.

[16] N. Ghani, P. Hancock, and D. Pattinson. Representations of stream processors using nested fixed points. *LMCS*, 5(3), 2009.

[17] E. Giménez. Codifying guarded definitions with recursive schemes. In *Selected Papers from the International Workshop on Types for Proofs and Programs*, TYPES '94, pages 39–59, London, UK, UK, 1995. Springer-Verlag.

[18] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types.* Cambridge University Press, New York, NY, USA, 1989.

[19] A. D. Gordon. Bisimilarity as a theory of functional programming. *Electronic Notes in Theoretical Computer Science*, 1:232–252, 1995.

[20] T. Hagino. *A Categorical Programming Language.* PhD, Univ. of Edinburgh, 1987.

[21] P. Hancock and A. Setzer. Interactive programs and weakly final coalgebras in dependent type theory. In L. Crosilla and P. Schuster, editors, *From sets and types to topology and analysis towards practicable foundations for constructive mathematics*, number 48 in Oxford Logic Guides, pages 115–134. Oxford University Press, 2005.

[22] R. Hasegawa. Categorical data types in parametric polymorphism. *Math. Struct. Comput. Sci.*, 4(01):71–109, 1994.

[23] D. Howe. Equality in lazy computation systems. In *, Fourth Annual Symposium on Logic in Computer Science, 1989. LICS '89, Proceedings*, pages 198–203, June 1989.

[24] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, pages 103–112, 1996.

[25] B. Jacobs. *Categorical Logic and Type Theory.* Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.

[26] B. Jacobs. Many-sorted coalgebraic modal logic: a model-theoretic study. *RAIRO - Theor. Inform. Appl.*, 35(01):31–59, Jan. 2001.

[27] B. Jacobs and J. Rutten. An introduction to (co)algebras and (co)induction. In D. Sangiorgi and J. Rutten, editors, *Advanced topics in bisimulation and coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2011.

[28] J. Kim. Higher-order algebras and coalgebras from parameterized endofunctors. *Electronic Notes in Theoretical Computer Science*, 264(2):141–154, Aug. 2010.

[29] B. Klin. Bialgebras for structural operational semantics: An introduction. *TCS*, 412(38):5043–5069, 2011.

[30] C. Kupke. *Finitary Coalgebraic Logics.* PhD thesis, Institute for Logic, Language and Computation, Amsterdam, 2006.

[31] P. Martin-Löf. About models for intuitionistic type theories and the notion of definitional equality. In *3rd Scandinavian Logic Symposium*, pages 81–109. North Holland and American Elsevier, 1975.

[32] R. Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, 4(1):1–22, Feb. 1977.

[33] R. E. Mogelberg. A type theory for productive coprogramming via guarded recursion. In T. A. Henzinger and D. Miller, editors, *Proceedings of LICS 2014*. ACM, 2014.

[34] K. Nakata and T. Uustalu. Resumptions, weak bisimilarity and big-step semantics for while with interactive i/o: An exercise in mixed induction-coinduction. *Electron. Proc. Theor. Comput. Sci.*, 32:57–75, Aug. 2010.

[35] A. M. Pitts. Parametric polymorphism and operational equivalence. *Math. Struct. Comput. Sci.*, 10(03):321–359, 2000.

[36] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science, pages 361–375. Springer Berlin Heidelberg, Jan. 1993.

[37] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, Dec. 1977.

[38] D. Pous. Complete lattices and up-to techniques. In Z. Shao, editor, *Programming Languages and Systems*, number 4807 in Lecture Notes in Computer Science, pages 351–366. Springer Berlin Heidelberg, 2007.

[39] D. Pous and D. Sangiorgi. Enhancements of the bisimulation proof method. In D. Sangiorgi and J. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction.* Cambridge University Press, 2012.

[40] M. Rößiger. Coalgebras and modal logic. *Electronic Notes in Theoretical Computer Science*, 33:294–315, 2000.

[41] J. Rot, F. Bonchi, M. Bonsangue, D. Pous, J. J. M. M. Rutten, and A. Silva. Enhanced coalgebraic bisimulation. *MSCS*, To Appear, 2014.

[42] J. J. M. M. Rutten. Universal coalgebra: A theory of systems. *Theor Comput Sci*, 249(1):3–80, Oct. 2000.

[43] J. J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *TCS*, 308(1-3):1–53, 2003.

[44] D. Sangiorgi. *Introduction to Bisimulation and Coinduction.* Cambridge University Press, New York, NY, USA, 2011.

[45] S. Staton. Relating coalgebraic notions of bisimulation. *Log. Methods Comput. Sci.*, 7(1), 2011.

[46] D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Logic in Computer Science, 1997. LICS'97. Proceedings., 12th Annual IEEE Symposium on*, pages 280–291. IEEE, 1997.

[47] N. Zeilberger. *The Logical Basis of Evaluation Order and Pattern-Matching.* PhD, Carnegie Mellon, Pittsburgh, 2009.

Index

A Proof of Confluence

In this appendix, we develop the details of the proof that $\longrightarrow$ is confluent on $\Lambda(A)$ for all types A (Thm. 2.10).

Recall from Def. 2.8 that $\Lambda(A)$ contains only well-covering terms, that is, terms in which all declaration bodies are well-covering with respect to $\lhd|$, a notion that was introduced in [3]. For the purpose of the proof, we introduce the rules of the covering relation $\lhd|$, which are a straightforward adaption of those given in loc. cit. The idea of these rules that we, starting at the empty sequence, build up a sequence of (typed) copatterns using these rules. In each step, we apply exactly all copatterns to cover a type or split a variable into all possible pattern cases, respectively.

Definition A.1. We say that a type A is *covered* by a sequence Q of copatterns if $A \lhd| Q$ can be derived from the following rules, where we write $Q\ ;\ Q'$ for the concatenation of two copattern sequences Q and Q'.

$$\frac{}{A \lhd| (\emptyset \vdash_{\text{cop}} \cdot : A \Rightarrow A)}\, C_{\text{Hole}} \qquad \frac{A \lhd| Q\ ;\ (\Gamma \vdash_{\text{cop}} q : A \Rightarrow A_1 \times A_2)}{A \lhd| Q\ ;\ (\Gamma \vdash_{\text{cop}} \pi_i\, q : A \Rightarrow A_i)_{i=1,2}}\, C_{\text{Prod}}$$

$$\frac{A \lhd| Q\ ;\ (\Gamma \vdash_{\text{cop}} q : A \Rightarrow (B \to C))}{A \lhd| Q\ ;\ (\Gamma, x : B \vdash_{\text{cop}} q\, x : A \Rightarrow C)}\, C_{\text{App}}$$

$$\frac{A \lhd| Q\ ;\ (\Gamma \vdash_{\text{cop}} q : A \Rightarrow \nu X.B)}{A \lhd| Q\ ;\ (\Gamma \vdash_{\text{cop}} \xi\ q : A \Rightarrow B[\nu X.B/X])}\, C_{\text{Out}}$$

$$\frac{A \lhd| Q\ ;\ (\Gamma, x : B_1 + B_2 \vdash_{\text{cop}} q : A \Rightarrow C)}{A \lhd| Q\ ;\ (\Gamma, x' : B_i \vdash_{\text{cop}} q[\kappa_i\, x'/x] : A \Rightarrow C)_{i=1,2}}\, C_{\text{Incl}}$$

$$\frac{A \lhd| Q\ ;\ (\Gamma, x : \mu X.B \vdash_{\text{cop}} q : A \Rightarrow C)}{A \lhd| Q\ ;\ (\Gamma, x' : B[\mu X.B/X] \vdash_{\text{cop}} q[\alpha\, x'/x] : A \Rightarrow C)}\, C_{\text{In}}$$

We show now that the reduction relation is confluent on Λ in three steps. First, we prove that (co)pattern matching is deterministic on well-covering terms. This allows us, as second step, to prove that $\longrightarrow^0_\Sigma$ that is confluent. Finally, we show, by induction, that $\longrightarrow^n_\Sigma$ is confluent for all n, thus $\longrightarrow = \bigcup_{n \in \mathbb{N}} \longrightarrow^n_\emptyset$ is confluent as all $\longrightarrow^n_\emptyset$ are disjoint.

We start by showing that copattern matching is deterministic.

Lemma A.2. *Let A be a type, Q a copattern sequence with $A \lhd| Q$ and e an evaluation context on A. If there exists a copattern q with $\Gamma \vdash_{\text{cop}} q : A \Rightarrow B$ in Q and contexts e_1, e_2 with $e = e_1[e_2]$, such that $q[\sigma] = e_2$, then q, e_1 and e_2 are unique with this property.*

Proof. Since any copattern sequence Q covering A is constructed using the rules in Def. A.1 starting at $Q_0 = (\emptyset \vdash_{\text{cop}} \cdot : A \Rightarrow A)$, we can proceed by induction in the application of said rules.

In the base case $Q = Q_0$, there is only one choice by definition, namely $q = \cdot$ and $e_1 = e$ and $e_2 = \cdot$.

So assume that for any Q we have a unique choice of q and $e = e_1[e_2]$. We make a case distinction on the rule used to construct Q' from Q. Note that we can distinguish two types of rules: C_{Prod}, C_{App} and C_{Out} increase the size of copatterns, whereas C_{Incl} and

C_{In} increase the size of patterns. We only prove the induction step for C_{App} and C_{Incl} as exemplary cases.

- Assume that Q' is constructed from $Q = Q''$; $(\Gamma \vdash_{\text{cop}} q : A \Rightarrow (B \to C))$ by an application of C_{App}, so that $Q' = Q''$; $(\Gamma, x : B \vdash_{\text{cop}} q\, x : A \Rightarrow C)$. Moreover, assume that q_0 in Q' matches e_2 for some splitting $e = e_1[e_2]$. If $q_0 \neq q\, x$, then $q_0 \in Q''$ and uniqueness of e_1, e_2 and q_0 follows by induction. Otherwise, if $q_0 = q\, x$, then by the typing of e we must have $e_2 = e_3\, t$ for some term $t : B$ and context e_3. Hence, we have that $q \in Q$ matches e_3 and, by induction, the splitting $e = e_1[e_3\, t]$ is unique, as the choice of q is. Combining these cases, we have that the splitting $e = e_1[e_2]$ and the choice of q_0 is still unique in Q'.
- Assume that Q' is constructed from $Q = Q''$; $(\Gamma, x : B_1 + B_2 \vdash_{\text{cop}} q : A \Rightarrow C)$ using the rule C_{Incl}, resulting in $Q' = Q''$; $(\Gamma, x' : B_i \vdash_{\text{cop}} q[\kappa_i\, x'/x] : A \Rightarrow C)_{i=1,2}$. If we now have a splitting $e = e_1[e_2]$ and a match $q[\kappa_i\, x'/x][\sigma] = e_2$, then we can define a substitution τ such that $q[\tau] = e_2$ by putting
$$\tau(y) = \begin{cases} \kappa_i\, \sigma(x'), & y = x \\ \sigma(y), & \text{otherwise} \end{cases}.$$
By the induction hypothesis, we now have that the splitting and q are unique for this match, thus the splitting and the choice of $q[\kappa_i\, x'/x]$ is unique. □

The next step is to prove confluence of the reduction in the base case, that is, of $\longrightarrow^0_\Sigma$. To do so, we invoke a result by Cirstea and Faure [10], which proves confluence of a reduction relation induced by a pattern matching algorithm for the so-called dynamic pattern λ-calculus. This calculus is an extension of (untyped) λ-calculus, in which λ-abstraction is allowed to have arbitrary terms in the abstraction, not just variable, that is to say, abstractions are of the for $\lambda M.N$ for arbitrary terms M and N. To interpret such an abstraction, we need to provide a *pattern matching algorithm*, which is a partial map from pairs of terms and sets of variables to substitutions, and is written as $\text{Sol}(M \ll N)$. Such a pattern matching algorithm induces a reduction relation by taking the parallel reduction closure of
$$(\lambda M.N)P \longrightarrow N[\sigma], \qquad \text{if } \sigma = \text{Sol}(M \ll P).$$
For more details, the reader should consult the corresponding paper.

The idea of how to encode the calculus we study in this paper into the dynamic pattern λ-calculus and the (co)pattern matching into a pattern matching algorithm is very simple. Since we are allowed to use arbitrary constants, we can encode all term constructors of our calculus directly, only that we need to turn the projections π_i and ξ into function arguments. For instance, $\pi_2(\xi\, t)$ becomes $M\ \xi\ \pi_2$, where M is the encoding of M. For a fixed declaration block Σ, the pattern matching algorithm is then the adaption of the matching with respect to evaluation context we used in Def. 2.6. This is similar to the case branching example given in [10].

The induced parallel reduction is not exactly the same as the compatible closure of contraction because they differ in the reduction of applications. However, the reduction relations can simulate each other, in the sense that if we can reduce a term, then the other relation can simulate this reduction in one or more steps. This is good enough to prove confluence: if parallel reduction is confluent for this encoding, then our reduction is confluent as well.

The heart of the matter are now the following three conditions from [10], which are sufficient to ensure that parallel reduction is confluent. Let us denote by $\mathrm{fv}(M)$ the free variables of M and by $\mathrm{dom}(\sigma)$ the domain of σ. Then the conditions are given by.

- H_0: If $\mathrm{Sol}(M \ll N) = \sigma$, then $\mathrm{fv}(M) = \mathrm{dom}(\sigma)$ and for all $x \in \mathrm{dom}(\sigma)$, we have $\mathrm{fv}(\sigma(x)) \subseteq \mathrm{fv}(N)$.
- H_1: Pattern matching commutes with substitution of variables not bound by the pattern: If $\mathrm{Sol}(M \ll N) = \sigma$ and $\mathrm{dom}(\tau) \cap \mathrm{fv}(M) = \emptyset$, then $\mathrm{Sol}(M \ll N[\tau]) = \tau \circ \sigma$.
- H_2: Pattern matching commutes with one-step reduction: If $\mathrm{Sol}(M \ll N) = \sigma$ and $N \longrightarrow N'$, then $\mathrm{Sol}(M \ll N) = \sigma'$ where σ' is the point-wise reduction of σ.

It is now straightforward to prove that the (co)pattern matching we used to define contraction fulfils these conditions.

Lemma A.3. *The (co)pattern matching on well-covering copattern sequences fulfils the conditions H_0, H_1 and H_2.*

Proof. Let A and Q be so that $A \lhd\!\mid Q$, and let $q[\sigma] = e$ for an evaluation context e and $q \in Q$. Note that q is unique by Lem. A.2.

H_0 Clearly, σ binds all variables in q and does not introduce fresh variables.

H_1 The condition H_1 requires, given a substitution τ with $\mathrm{dom}(\tau) \cap \mathrm{fv}(q) = \emptyset$, that $q[\sigma][\tau] = e[\tau]$. This is clearly the case, as no variable in q are substituted.

H_2 Assume we have $e \longrightarrow e'$, we need to show that $q[\sigma'] = e'$ with $\sigma \longrightarrow \sigma'$, where we can use the same q by uniqueness. Here we use the obvious lifting of $\longrightarrow$ to evaluation contexts and substitutions. If e was closed e' is still closed and by Lem. A.2 we still have a match $q[\sigma'] = e'$. The only interesting case to show that $\sigma \longrightarrow \sigma'$ is $q = x$, i.e. $x[t'/x] = t'$, but since $t \longrightarrow t'$ we clearly have $\sigma = [t/x] \longrightarrow [t'/x] = \sigma'$. The rest follows by induction in q. □

By the above discussion, confluence follows on terms without **rlet**-bindings from [10].

Lemma A.4. *On $\Lambda^{\Sigma}(A)$, $\longrightarrow^{0}_{\Sigma}$ is confluent for all well-covering Σ.*

Lemma A.4 is the base case for the main result, the confluence of $\longrightarrow^{k}_{\Sigma}$ for any k, which we prove by induction. To justify that this induction is actually well-formed, we need the following technical lemma. Let us denote the **rlet**-nesting depth of any syntactic entity S by $d(S)$, where S can be a term, a declaration block etc.

Lemma A.5. *For all terms t, t' with $t \longrightarrow_{\Sigma} t'$, we have $d(t') \leq \max\{d(t), d(\Sigma)\}$.*

Proof. Let t and t' with $t \longrightarrow_{\Sigma} t'$, and note that we then have that $t \longrightarrow^{d(t)}_{\Sigma} t'$, by definition of $\longrightarrow_{\Sigma}$. We observe that the only possibility to change the **rlet**-nesting is a use of contraction $e[f] \succ_{\Sigma'} r$, where $e[f]$ is a subterm of t.

There are now two possibilities: Either there there is a declaration block $\Sigma'' \subseteq \Sigma'$ that contains f and a term s that contains $e[f]$ as a subterm, such that $\mathbf{rlet}\,\Sigma''\,\mathbf{in}\,s$ is a subterm of t, or f is already contained in Σ.

In the first case, we reduce the subterm $\mathbf{rlet}\,\Sigma''\,\mathbf{in}\,s$ to $\mathbf{rlet}\,\Sigma''\,\mathbf{in}\,s'$, which induces the reduction $t \longrightarrow_{\Sigma} t'$ by the compatible closure and the **rlet**-rules. In turn, this reduction of subterms must be given by a reduction $s \longrightarrow_{\Sigma'} s'$, where $\Sigma'' \subseteq \Sigma'$, which is then induced by a contraction $e[f] \succ_{\Sigma'} r$ with $(f : A = D) \in \Sigma''$. By definition, we now have $d(\mathbf{rlet}\,\Sigma''\,\mathbf{in}\,s) = \max\{d(\Sigma'') + 1, d(s)\}$, thus $d(r) \leq d(D)$ and $d(s') \leq \max\{d(r), d(s)\} \leq \max\{d(D), d(s)\}$. This implies that $d(\mathbf{rlet}\,\Sigma''\,\mathbf{in}\,s') = \max\{d(\Sigma'') + 1, d(s')\} \leq \max\{d(\Sigma'') + 1, d(D), d(s)\} = d(\mathbf{rlet}\,\Sigma''\,\mathbf{in}\,s)$, where the last step follows from $(f : A = D) \in \Sigma''$. Since the only change caused by the reduction $t \longrightarrow t'$ happens in s, we have $d(t') \leq d(t)$.

In the second case, we similarly get that $d(r) \leq d(\Sigma)$. Together with the first case, we have that $d(r) \leq \max\{d(t), d(\Sigma)\}$. □

This result allows us to prove that $\longrightarrow^k_\Sigma$ is confluent using induction in k, as the nesting depth cannot be increased by reduction steps.

Proof of Theorem 2.10. We show that $\longrightarrow^k_\Sigma$ is confluent by induction in k, which implies that $\longrightarrow_\Sigma$ is confluent because every term has a unique **rlet**-depth. This induction is well-founded by Lem. A.5. The base case is dealt with in Lem. A.4, so we immediately continue with the induction step.

Assume that $\longrightarrow^k_\Sigma$ is confluent for any well-covering Σ, we show that for any well-covering Σ the relation $\longrightarrow^{k+1}_\Sigma$ confluent. As usual, we show that for terms t, t_1 and t_2 with $t \longrightarrow^{k+1}_\Sigma t_i$ there is a term t_3 with $t_i \twoheadrightarrow^{k+1}_\Sigma t_3$, which is called weak confluence and implies confluence. First, we note that if the reductions to t_1 and t_2 both come from the compatible closure, then we can find t_3 by induction in the definition of the compatible closure. The base case of this induction requires the existence of t_3 for the case where $t \longrightarrow^{k+1}_\Sigma t_1$ and $t \succ^{k+1}_\Sigma t_2$, which we prove in the following.

i) If $t = \mathbf{rlet}\,\Sigma'\,\mathbf{in}\,s$, then we have the following cases.

   (a) $t_i = \mathbf{rlet}\,\Sigma'\,\mathbf{in}\,s_i$ with $s \longrightarrow^k_{\Sigma,\Sigma'} s_i$. Since $\longrightarrow^k_{\Sigma,\Sigma'}$ is confluent by induction, there is an s_3 with $s_i \longrightarrow^k_{\Sigma,\Sigma'} s_3$ and we can join t_1 and t_2 with $\mathbf{rlet}\,\Sigma\,\mathbf{in}\,s_3$.

   (b) $t_1 = \mathbf{rlet}\,\Sigma'\,\mathbf{in}\,s_1$ with $s \longrightarrow^k_{\Sigma,\Sigma'} s_1$, $s = \alpha\,s'$ and $t_2 = \alpha(\mathbf{rlet}\,\Sigma'\,\mathbf{in}\,s')$. Then we must have that $s_1 = \alpha\,s'_1$ with $s_1 \longrightarrow^k_{\Sigma,\Sigma'} s'_1$, hence we can t_1 and t_2 by

$$\begin{tikzcd}
 & t = \mathbf{rlet}\,\Sigma'\,\mathbf{in}(\alpha\,s') \arrow[dl] \arrow[dr] & \\
t_1 = \mathbf{rlet}\,\Sigma'\,\mathbf{in}(\alpha\,s'_1) \arrow[dr] & & t_2 = \alpha(\mathbf{rlet}\,\Sigma'\,\mathbf{in}\,s') \arrow[dl] \\
 & \alpha(\mathbf{rlet}\,\Sigma'\,\mathbf{in}\,s'_1) &
\end{tikzcd}$$

   (c) We proceed analogously if $s = \kappa_i s'$.

   (d) The other cases follow by symmetry.

ii) If $t = \xi(\mathbf{rlet}\,\Sigma'\,\mathbf{in}\,s)$, $t_2 = \mathbf{rlet}\,\Sigma'\,\mathbf{in}(\xi\,s)$ and $t_1 = \xi(\mathbf{rlet}\,\Sigma'\,\mathbf{in}\,s_1)$ with $s \longrightarrow^k_{\Sigma,\Sigma'} s_1$, then we can reduce t_1 to $\mathbf{rlet}\,\Sigma'\,\mathbf{in}(\xi\,s_1)$ and s to s_1. Thus the joining term is $\mathbf{rlet}\,\Sigma'\,\mathbf{in}(\xi\,s_1)$.

iii) We proceed analogously if t is an **rlet** in context of an application or π_i.

iv) The remaining cases are either trivial because the same reduction happens on both t_1 and t_2, they follow by symmetry, or combinations of reductions by $\succ_{\Sigma}^{k+1}$ are excluded by the types of t_1 and t_2.

This proves that, if $t \longrightarrow_{\Sigma}^{k+1} t_1$ and $t \succ_{\Sigma}^{k+1} t_2$, then there exists t_3 with $t_i \twoheadrightarrow_{\Sigma}^{k+1} t_3$. It is straightforward to extend this by induction to the compatible closure, hence to arbitrary reductions towards t_2. This shows that $\longrightarrow_{\Sigma}^{k+1}$ is confluent for any well-covering Σ, provided $\longrightarrow_{\Sigma'}^{k}$ is for any well-covering Σ'. Thus, by induction in k, $\longrightarrow_{\Sigma}$ is confluent. □