ELSEVIER

Available online at www.sciencedirect.com
SCIENCE DIRECT®

Theoretical Computer Science 360 (2006) 415–439

Theoretical Computer Science

www.elsevier.com/locate/tcs

Interfaces as functors, programs as coalgebras—A final coalgebra theorem in intensional type theory

Markus Michelbrink*, 1

Department of Computer Science, University of Wales Swansea, Singleton Park, Swansea, SA2 8PP, UK

Received 24 May 2005; received in revised form 9 March 2006; accepted 16 May 2006

Communicated by B.P.F. Jacobs

Abstract

In [P. Hancock, A. Setzer, Interactive programs in dependent type theory, in: P. Clote, H. Schwichtenberg (Eds.), Proc. 14th Annu. Conf. of EACSL, CSL'00, Fischbau, Germany, 21–26 August 2000, Vol. 1862, Springer, Berlin, 2000, pp. 317–331, URL ⟨citeseer.ist.psu.edu/article/hancock00interactive.html⟩; P. Hancock, A. Setzer, Interactive programs and weakly final coalgebras in dependent type theory, in: L. Crosilla, P. Schuster (Eds.), From Sets and Types to Topology and Analysis. Towards Practicable Foundations for Constructive Mathematics, Oxford Logic Guides, Clarendon Press, 2005, URL ⟨www.cs.swan.ac.uk/∼csetzer/⟩] Hancock and Setzer introduced rules to extend Martin-Löf's type theory in order to represent interactive programming. The rules essentially reflect the existence of weakly final coalgebras for a general form of polynomial functor. The standard rules of dependent type theory allow the definition of inductive types, which correspond to initial algebras. Coalgebraic types are not represented in a direct way. In this article we show the existence of final coalgebras in intensional type theory for these kind of functors, where we require uniqueness of identity proofs (UIP) for the set of states S and the set of commands C which determine the functor. We obtain the result by identifying programs which have essentially the same behaviour, viz. are bisimular. This proves the rules of Setzer and Hancock admissible in ordinary type theory, if we replace definitional equality by bisimulation. All proofs [M. Michelbrink, Verifications of final coalgebra theorem in: Interfaces as Functors, Programs as Coalgebras—A Final Coalgebra Theorem in Intensional Type Theory, 2005, URL ⟨www.cs.swan.ac.uk/∼csmichel/⟩] are verified in the theorem prover agda [C. Coquand, Agda, Internet, URL ⟨www.cs.chalmers.se/∼catarina/agda/⟩; K. Peterson, A programming system for type theory, Technical Report, S-412 96, Chalmers University of Technology, Göteborg, 1982], which is based on intensional Martin-Löf type theory.
© 2006 Elsevier B.V. All rights reserved.

MSC: 03B15; 03F65; 03B70; 16W30; 18A15; 18D15; 68Q60; 68Q85

Keywords: Dependent type theory; Interactive programming; Coalgebra

1. Introduction

Martin-Löf type theory [28,35] is a very carefully developed framework for constructive mathematics. It is well suited as a theory for program construction since it is possible to express both specification and programs within the

* Tel.: +44 1792 295393; fax: +44 1792 295708.
E-mail address: m.michelbrink@swansea.ac.uk.
1 Supported by EPSRC Grant GR/S30450/01.

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.05.033

same formalism. Types in Martin-Löf type theory can be seen as program specifications via the proposition-as-types interpretation [34,6,37]. Inhabitants of these types are programs which fulfil the required specification. Running such a program means to evaluate an expression. One of the design features of the framework is that the evaluation of a well-typed program always terminates. Further, there is no interaction with the environment. In order to introduce interaction into type theory and to allow the non-termination of programs, Hancock and Setzer [17,19] introduced the notions of (state dependent) interfaces and interactive programs. Their approach results in an extension of type theory by rules expressing the existence of weakly final coalgebras for the functors determined by interfaces. This coalgebraic rules give a comfortable way to reason about interactive programs. However, coalgebraic types are not represented directly in standard type theory. In fact, they are classical examples of impredicative conceptions whereas Martin-Löf type theory is a strictly predicative theory. Predicative type theories play a particular role for giving foundational interpretations of programming languages. They have multiple mathematical models, notably set theoretic, PER models and denotational models, that provide precise definitions of programming language features, due to their explicit inductive construction.

On the other side one has to be careful adding rules to type theory. That this may have disastrous consequences can be seen e.g. in Martin-Löf's Mathematics of Infinity [29] where it is shown that type theory becomes inconsistent when the formal laws for the fixed point operator are adjoined to it.

However, in this work we show that it is possible to reason about interactive programs in standard predicative type theory as long as we replace the definitional equality in the rules [17,19] by bisimulation. This is done by constructing final coalgebras for the functors mentioned above. The basic idea for this construction is essentially the same as for the model construction in Michelbrink/Setzer [34]. However, the proof that there is a final coalgebra for this kind of functors is surprisingly hard. This is due to the fact that we work in intensional type theory, where we have to deal with the problem that types depending on propositionally equal elements may not be equal. However, unlike the extensional version intensional type theory has a number of desirable features we do not want to miss: all well-typed expressions normalize and well-typedness, type-hood, type-checking as well as definitional equality are decidable.

The theory of types developed by Martin-Löf "is intended to be a full scale system for formalizing intuitionistic mathematics" [30]. As a foundational theory it is thought to be open-ended, in the sense that we might extend it by rules for new types provided the informal semantic principles of the theory are respected. In this article we work with an extension of Martin-Löf type theory that accommodates inductive–recursive definitions. A first example of simultaneous induction–recursion is Martin-Löf's definition of the first universe á la Tarski [28]. The general schema for this kind of definition is introduced and investigated by Dybjer [10].

The paper is organized as follows. In Section 2 we restate the original definition of interfaces and programs, try to explain the concept of intensional identity, the meaning it has for constructive reasoning and describe the difficulties which arise using this concept. We discuss families and predicates and how they are related and give a new modified definition of interfaces. In Section 3 we introduce our category and in the following Section 4 the endofunctor Prog on this category, for which we are going to show that there is a final coalgebra in the category. In Section 5 we define a coalgebra for this functor, which consist in a family of sets CT, equivalence relations on this sets and a morphism $\mathsf{elim} \colon \mathsf{CT} \to \mathsf{Prog}\ \mathsf{CT}$. In Section 6 we introduce the unique morphism. However, to prove that the function defined indeed belongs to the category and that it is the unique morphism making the coalgebra square commute we have to do some more work. In Section 7 we define the repetition of the unique morphism and prove our Main Lemma. The Main Lemma is then used to prove that the morphism defined in Section 6 belongs to the category (is extensional) and is the unique morphism making the diagram commute. In Section 8 we point out how to get a final coalgebra for the original functor of Hancock/Setzer from this. In Section 10 we conclude by describing some future and related work.

We use the following notations: $t \rightsquigarrow t'$ for t evaluates to t', $t \leftrightsquigarrow t'$ for t, t' evaluate to the same value, A for the type A is inhabited, $id : t \doteq t'$ or $id : t \doteq_A t'$ for id is an inhabitant of the identity type. We use the notation $(x : A) \to B\ x$ for the product type and $\mathsf{sig}\ \mathsf{m}_0 : A_0, \ldots, \mathsf{m}_n : A_n\ \mathsf{m}_0 \ldots \mathsf{m}_{n-1}$ for sigma types where the components of $a : \mathsf{sig}\ \mathsf{m}_0 : A_0, \ldots, \mathsf{m}_n : A_n\ \mathsf{m}_0 \ldots \mathsf{m}_{n-1}$ are accessed via a_{m_i} for $i = 0, \ldots, n$. We denote the canonical elements of the sigma types by $\langle a_0, \ldots, a_n \rangle$ and abbreviate $\mathsf{sig}\ \mathsf{fst} : A,\ \mathsf{snd} : B\ \mathsf{fst}$ by $\sum(A, B)$ or $\sum(x : A.B\ x)$ to emphasize x. The sentential connectives $\forall, \exists, \wedge, \vee, \Rightarrow$ for this type of constructors are used in the standard way to emphasize the reading of types as propositions. We sometimes suppress arguments which can be inferred from other arguments, for instance we write $\mathsf{subst}\ id\ b$ instead of $\mathsf{subst}\ A\ B\ a\ a'\ id\ b$. We also use the notation _ for missing arguments. We use the notations False and True for the type with zero and one canonical element, respectively. To improve readability we overload some function symbols, e.g. st, co. However, functions denoted by equal symbols have equal codomains whereas the argument types may be different.

2. Basic definitions and concepts

2.1. Interfaces and interactive programs

In [17] Hancock and Setzer give the following definition of an interface:

An *interface* is a quadruple (S, C, R, n) s.t.

- $S : \mathsf{Set}$
- $C : S \to \mathsf{Set}$
- $R : (s : S, C\ s) \to \mathsf{Set}$
- $n : (s : S, c : C\ s, R\ s\ c) \to S.$

The elements of the set S are called states, $C\ s$ is the set of commands in state $s : S$, $R\ s\ c$ the set of responses to a command $c : C\ s$ in state $s : S$, and $n\ s\ c\ r$ the next state of the system after this interaction.

A *program* for this interface starting in state $s : S$ is a quadruple $(A, \mathsf{c}, \mathsf{next}, a)$ s.t.

- $A : S \to \mathsf{Set}$
- $\mathsf{c} : (s : S, A\ s) \to C\ s$
- $\mathsf{next} : (s : S, a : A\ s, r : R\ s\ (\mathsf{c}\ s\ a)) \to A\ (n\ s\ (\mathsf{c}\ s\ a)\ r)$
- $a : A\ s.$

The elements of the set $A\ s$ are understood as programs starting in the state s. The command $\mathsf{c}\ s\ a$ is the command issued by the program $a : A\ s$, and $\mathsf{next}\ s\ a\ r$ is the program that will be executed, after having obtained for command $\mathsf{c}\ s\ a$ the response $r : R\ s\ (\mathsf{c}\ s\ a)$. The execution of a program $a : A\ s$ proceeds as follows. First we compute $\mathsf{c}\ s\ a$ and issue this command. Then we wait for a response $r : R\ s\ (\mathsf{c}\ s\ a)$ from the real world. When we have obtained a response r we compute the new program $\mathsf{next}\ s\ a\ r$. This cycle is repeated until we reach a command c with no responses. It may be undecidable if this is the case. It should also be noted that a program may wait forever for a response. See [17] for further motivations.

Note that in the above definition programs are given by arbitrary families of sets $A : S \to \mathsf{Set}$. That means the whole range of sets can be used to introduce elements into the set of all programs. In particular, the set of programs itself may be used. This is a violation of the vicious-circle principle: impredicative definitions should not be used. That is, an object should not be defined in terms of a totality to which the object itself belongs. In other words, no totality can contain members defined in terms of itself. The vicious-circle principle is taken very seriously in Martin-Löf type theory.

If we combine $\mathsf{c}\ s\ a$ and $\mathsf{next}\ s\ a$ we get an element of $\mathsf{Prog}_{\mathsf{HS}}\ A\ s := \sum(c : C\ s.(r : R\ s\ c) \to A\ (n\ s\ c\ r))$. Since there is no way to get the set of all programs directly in a predicative framework, Hancock and Setzer expanded Martin-Löf type theory. This results in a type theory where the adjoined rules express the existence of a (weakly) final coalgebra for the functor $\mathsf{Prog}_{\mathsf{HS}}$.

We are going to show that under certain assumptions on the sets of states and commands the existence of this set of programs can be proved in ordinary type theory. The proof is surprisingly hard. The reason for this is that we work in intensional type theory.

2.2. Intensional identity

Under the proposition-as-types interpretation, propositions are nothing other than types. That a proposition is true means that the type is inhabited. In order to have an internal representation of equality, identity types are introduced. The main purpose of these identity types is to be able to make the assumption that two objects of a type are identical, i.e. to express identity of objects on the left side of an implication. Martin-Löf type theory can be formulated on top of a theory of logical types (logical framework) [35]. This is a typed $\lambda\beta\eta$-calculus with dependent function types, a special type Set and a rule which states that each object of Set is also a type. Sets are given by formation, introduction, elimination and equality rules. The formation rules say how to build sets, the introduction rules say what the canonical elements of the set are. Elimination and equality rules say how to eliminate set formers. β- and η-conversion together with the equality rules give definitional equality.

There are two main versions of Martin-Löf type theory: extensional and intensional type theory. The difference lies in the treatment of the identity type. In both versions the formation and introduction rules of the identity type are the same:

$$\frac{A : \mathsf{Set} \quad a, b : A}{a \doteq_A b : \mathsf{Set}} \qquad \frac{A : \mathsf{Set} \quad a : A}{\mathsf{refl}\ a : a \doteq_A a}$$

The difference is in the elimination and equality rules for the identity type. The elimination rules in extensional type theory identify propositional and definitional identity:

$$\frac{p : a \doteq_A b}{a = b : A}$$

This renders type theory undecidable, i.e. well-typedness, type-checking, type-hood and definitional equality become undecidable [22]. This is in contrast to intensional type theory. There is a deep symmetry between the introduction rules on the one side and the elimination and equality rules on the other side in intensional type theory. The elimination rules for all sets can be understood as structural induction rules: a proposition is true for all elements iff the proposition is true for the canonical elements of the set. In fact, elimination and equality rules can be calculated from the introduction rules [9]. This holds as well for the identity type:

$$\frac{\begin{array}{l} C : (x, y : A, p : x \doteq_A y) \to \mathsf{Set} \\ c : (x : A) \to C\ x\ x\ (\mathsf{refl}\ x) \qquad a, b : A \quad p : a \doteq_A b \end{array}}{\mathsf{idpeel}\ C\ c\ a\ b\ p : C\ a\ b\ p}$$

with equality $\mathsf{idpeel}\ C\ c\ a\ a\ (\mathsf{refl}\ a) = c\ a$. Surprisingly this very weak elimination rule allows to deduce the usual properties of equality, notably Leibniz' principle ($C\ a$ implies $C\ b$ for $a \doteq b$). However, working with intensional identity becomes very awkward. The reason for this is that propositional and definitional equality do not collapse. That is, two instances of a type family with indices which are not convertible, just propositionally equal, are not the same type, i.e. $c : C\ a$ is in general not an element of $C\ b$ if a equals b, though if $p : a \doteq b$ and $c : C\ a$ we get an element $\mathsf{subst}\ p\ c : C\ b$. The trouble is that this element depends on the proof p and there is no general way to conclude that $\mathsf{subst}\ p\ c$ equals $\mathsf{subst}\ q\ c$ for $p, q : a \doteq b$.

We frequently use the following well known (and easy to prove) principles:

Principle 1.

$$a_0 \doteq a_1 \Rightarrow f\ a_0 \doteq f\ a_1$$

for $A, B : \mathsf{Set}$, $f : A \to B$, $a_0, a_1 : A$.

Principle 2.

$$\langle a_0, b_0 \rangle \doteq_{\sum(A,B)} \langle a_1, b_1 \rangle \quad \Leftrightarrow \quad a_0 \doteq_A a_1 \wedge \bar{b}_0 \doteq_{B\, a_1} b_1$$

for $A : \mathsf{Set}$, $B : A \to \mathsf{Set}$, $a_i : A$, $b_i : B_i$, $i = 0, 1$ *and* $\bar{b}_0$ *obtained from* b_0 *by the inhabitant of* $a_0 \doteq a_1$.

2.3. Families and predicates

What makes type theory into dependent type theory is that types may depend on elements of other types. A family of sets is given by a set *IndexP* and a function $P : \mathit{IndexP} \to \mathsf{Set}$. The function P may as well be seen as a predicate on *IndexP*. On the other hand it is often technically simpler to work with a more fibration-like view of families: a family is given by two sets *CoIndexF*, *IndexF* and a function $F : \mathit{CoIndexF} \to \mathit{IndexF}$. We call the former predicate and the latter family. It is possible to switch between these notions in the following ways: from predicate P to family F (pr_0 denotes the first projection):

$$\begin{aligned} \mathit{CoIndexF} &:= \textstyle\sum(\mathit{IndexP}, P) \\ \mathit{IndexF} &:= \mathit{IndexP} \\ F &:= \mathrm{pr}_0 \ : \ \textstyle\sum(\mathit{IndexP}, P) \to \mathit{IndexP}. \end{aligned}$$

From family F to predicate P define $IndexP := IndexF$ and let $P\ i$ be given by the following rules:

$$\begin{array}{ccc}
\text{Formation} & \text{Introduction} & \text{Elimination} \\
\dfrac{i : IndexF}{P\ i : \mathsf{Set}} & \dfrac{c : CoIndexF}{\mathsf{intro}\ c : P\ (F\ c)} & \dfrac{\begin{array}{c} i : IndexF \quad c' : P\ i \\ B : (i : IndexF, c' : P\ i) \to \mathsf{Set} \\ b : (c : CoIndexF) \to B\ (F\ c)\ (\mathsf{intro}\ c)\end{array}}{\mathsf{elim}\ B\ b\ i\ c' : B\ i\ c'}
\end{array}$$

where $\mathsf{elim}\ B\ b\ (F\ c)\ (\mathsf{intro}\ c)$ evaluates to $b\ c$. Note that the latter gives exactly the rules for intensional identity if we take as family $\Delta : A \to A \times A$ with $\Delta\ a := (a, a)$. We write $\mathsf{PredToFam}\ P$ and $\mathsf{FamToPred}\ F$, respectively, for the predicate and family we gain by the way above. Intuitively we can think about $\mathsf{FamToPred}\ F$ as the pre-image function F^{-1}.

We say that $f : A \to B$ is a bijection iff there is a $g : B \to A$ such that $a \doteq (g(f\ a))$ and $b \doteq (f(g\ b))$ are inhabited for all $a : A, b : B$. We write $A \simeq B$ iff there is such a bijection. It is easy to establish the following bijections:

$$P\ i \simeq \mathsf{FamToPred}\ (\mathsf{PredToFam}\ P)\ i$$

$$\mathrm{iso} : CoIndexF \simeq (\mathsf{PredToFam}(\mathsf{FamToPred}\ F))_{\mathsf{CoIndexF}}.$$

In the second case the functions $\mathrm{pr}_0 \circ \mathrm{iso} = (\mathsf{PredToFam}(\mathsf{FamToPred}\ F)) \circ \mathrm{iso}$ and F are pointwise equal.

There is a second approach to get a predicate P from a family F. This approach uses the identity set: define $IndexP := IndexF$ and

$$P\ i := \textstyle\sum(c : CoIndexF, (F\ c) \doteq i)$$

for $i : IndexF$. We write $\mathsf{FamToPred}'\ F$ for this predicate. Again it is not too hard to establish the following bijections:

$$P\ i \simeq \mathsf{FamToPred}'\ (\mathsf{PredToFam}\ P)\ i$$

$$iso : CoIndexF \simeq (\mathsf{PredToFam}\ (\mathsf{FamToPred}'\ F))_{\mathsf{CoIndexF}}$$

and to prove that in the second case the functions $\mathrm{pr}_0 \circ \mathrm{iso} = (\mathsf{PredToFam}\ (\mathsf{FamToPred}'\ F)) \circ \mathrm{iso}$ and F are pointwise equal. Note that the index set stays the same all the time and that

$$\mathsf{FamToPred}\ (\mathsf{PredToFam}\ P)\ i \simeq \mathsf{FamToPred}'\ (\mathsf{PredToFam}\ P)\ i$$

$$(\mathsf{PredToFam}\ (\mathsf{FamToPred}\ F))_{\mathsf{CoIndex}} \simeq (\mathsf{PredToFam}\ (\mathsf{FamToPred}'\ F))_{\mathsf{CoIndexF}}.$$

This is a little bit remarkable since the second approach seems to multiply elements due to the fact that there may be more than one inhabitant of $(F\ c) \doteq i$. The phenomenon is related to the fact that we can prove

$$\mathsf{Collapse}\ \textstyle\sum(a : A, a \doteq a')$$

for $a' : A$ but in general not

$$\mathsf{Collapse}\ (a \doteq a')$$

for $a, a' : A$ where $\mathsf{Collapse}\ A$ is $\forall a, a' : A.a \doteq a'$.

2.4. A simpler definition of interfaces

What makes work with the above interface definition clumsy is that there are too many dependencies. The commands depend on the states, the responses on the commands and the next state on the state, the command and the response. This seems to be redundant since the information to which state a command belongs should already be given by the command itself and similarly for the responses and the next state. Hence the responses should depend only on the command and the next state on the response. The way to achieve this is to work with families instead of predicates:

Definition 3 (*Interface*). An interface is given by sets $\mathsf{S}, \mathsf{C}, \mathsf{R}$ and functions $\mathsf{st} : \mathsf{C} \to \mathsf{S}$, $\mathsf{co} : \mathsf{R} \to \mathsf{C}$, $\mathsf{nxt} : \mathsf{R} \to \mathsf{S}$.

Given an interface (S, C, R, n) in the sense of Hancock/Setzer we get an interface in the new sense by

$$\mathsf{st} := \mathsf{PredToFam}\ C$$

$$\mathsf{co} := \mathsf{PredToFam}\ R'$$

and setting

$$\mathsf{nxt}(((s, c), r)) := n\ s\ c\ r,$$

where R' is the uncurried version of R. The altered definition determines a functor (see Section 4). We are going to prove that this functor has a final coalgebra and use this result to get a final coalgebra for the original functor of Hancock/Setzer above. However, we have not succeeded to prove the result in its most general form for arbitrary sets S, C. In order for the proof to go through we need a principle known as uniqueness of identity proofs on the sets S, C. This principle states that all the inhabitants of $a \doteq a'$ are identical, that is,

$$\forall a, a' : A.\ \mathsf{Collapse}\ (a \doteq a').$$

We write UIP A for $\forall a, a' : A.\ \mathsf{Collapse}\ (a \doteq a')$. As shown by Hofmann [21,22] UIP A is not provable for arbitrary sets A. However, it is provable for the enumeration types, the natural number type and preserved by the identity type and the sum type constructors [21], that is,

$$\mathsf{UIP}\ A \Rightarrow \forall a, a' : A.\mathsf{UIP}\ (a \doteq_A a')$$

and

$$\mathsf{UIP}\ A \Rightarrow (\forall a : A.\mathsf{UIP}\ B\ a) \Rightarrow \mathsf{UIP}\ \textstyle\sum(A, B).$$

More general UIP A follows from decidability of identity [20] that is

$$\forall a, a' : A.(a \doteq_A a') \vee (a \not\doteq_A a'),$$

which is also preserved by the sum type constructor. Streicher [42] noticed that UIP A is provable if in the elimination rules for the above identity type the type of C is changed from $(x, y : A, p : x \doteq_A y) \to \mathsf{Set}$ to $(x : A, p : x \doteq_A x) \to \mathsf{Set}$. Using this elimination rule is equivalent to pattern matching [31], which therefore proves UIP as well. However, in this case elimination cannot be justified as structural induction. In the following we assume UIP for the sets S and C.

3. The category of S-indexed families of setoids

We are going to define the category of S-indexed families of setoids. The ambient category of setoids is a model of intensional type theory [21]. The set of states S determines the following (presheaf-)category: objects are triples

$$X : \mathsf{S} \to \mathsf{Set}$$

$$\equiv_X : (s : \mathsf{S}, X\ s, X\ s) \to \mathsf{Set}$$

$$\mathsf{eq}_X : (s : \mathsf{S}) \to \mathsf{equivalence}\ (\equiv_X s),$$

where equivalence R says that R is an equivalence (reflexive, transitive, symmetric) relation.

We use the notations $\equiv$, $\equiv_X$ and $\equiv_s$ for the binary relation $(s : \mathsf{S})$

$$\equiv_X s \subseteq X\ s \times X\ s.$$

We say $\equiv_X : (s : \mathsf{S}) \to X\ s \to X\ s \to \mathsf{Set}$ is an equivalence relation iff all relations $\equiv_s \subseteq X\ s \times X\ s$ are equivalence relations. Morphism $f : (X, \equiv_X, \mathsf{eq}_X) \to (Y, \equiv_Y, \mathsf{eq}_Y)$ are given by a family of S-indexed *extensional* functions in the sense that

$$f : (s : \mathsf{S}) \to X\ s \to Y\ s$$

and

$$x \equiv_X x' \Rightarrow f\ s\ x \equiv_Y f\ s\ x'$$

for $s : \mathsf{S}, x, x' : X\ s$. We use the same notation for the morphism and the function f. If we want to emphasize the relations $\equiv_X, \equiv_Y$ we sometimes say that f is $(\equiv_X, \equiv_Y)$-extensional. We identify $f, g : (X, \equiv_X, \mathsf{eq}_X) \to (Y, \equiv_Y, \mathsf{eq}_Y)$ iff

$$x \equiv_X x' \Rightarrow f\ s\ x \equiv_Y g\ s\ x'$$

for all $s : \mathsf{S}, x, x' : X\ s$.

It is easily verified that this gives a category.

4. The endofunctor Prog

The interface (S, C, R, st, co, nxt) determines the endofunctor Prog given by

$$\begin{aligned}
&\mathsf{Prog}\ X\ s : \mathsf{Set}\\
=\quad &\mathsf{sig\ command} : \mathsf{C}\\
&\quad \mathsf{id}^{\mathsf{S}}_{\mathsf{co}} : (\mathsf{st}\ c) \doteq s\\
&\quad \mathsf{next}_{\mathsf{El}} : (r : \mathsf{R},\ (\mathsf{co}\ r) \doteq c) \to X(\mathsf{nxt}\ r)
\end{aligned}$$

for $X : \mathsf{S} \to \mathsf{Set}$ with equivalence relation

$$\begin{aligned}
&\mathsf{Prog} \equiv_X s\ \langle c_0, ids_0, f_0\rangle\ \langle c_1, ids_1, f_1\rangle : \mathsf{Set}\\
=\quad &\mathsf{sig\ idc} : c_0 \doteq c_1\\
&\quad \mathsf{fct} : (r : \mathsf{R}, idcr : (\mathsf{co}\ r) \doteq c_0) \to f_0\ r\ idcr \equiv_{(\mathsf{nxt}\ r)} f_1\ r\ idcr',
\end{aligned}$$

where $idcr' := \mathsf{subst\ idc}\ idcr$. We use the notation $\equiv_{\mathsf{Prog}}$ for this relation. By some simple calculations it follows that $\equiv_{\mathsf{Prog}}$ is an equivalence relation if $\equiv$ is an equivalence relation. We allow some abuse of notations. Prog takes a family of sets $X : \mathsf{S} \to \mathsf{Set}$, an equivalence relation $\equiv_X$ on X and a witness for the fact that $\equiv_X$ is an equivalence relation and gives a triple consisting of a family of sets $\mathsf{Prog}\ X : \mathsf{S} \to \mathsf{Set}$ an equivalence relation $\mathsf{Prog}\ \equiv_X$ on $\mathsf{Prog}\ X$ and a corresponding witness.

The morphism part of the functor Prog is given by

$$\begin{aligned}
\mathsf{Prog}\ g\ s\ &:\ \mathsf{Prog}\ X\ s \to \mathsf{Prog}\ Y\ s\\
\mathsf{Prog}\ g\ s\ \langle c, ids, f\rangle &= \langle c, ids, \lambda r : \mathsf{R}, idc : (\mathsf{co}\ r) \doteq c.\ g\ (\mathsf{nxt}\ r)\ (f\ r\ idc)\rangle.
\end{aligned}$$

If g is extensional then $\mathsf{Prog}\ g$ is extensional too. To see this, let

$$\langle c_0, ids_0, f_0\rangle \equiv_{\mathsf{Prog}} \langle c_1, ids_1, f_1\rangle.^{2}$$

Then we have $idc : c_0 \doteq c_1$. Let $r : \mathsf{R}$ and $idcr : (\mathsf{co}\ r) \doteq c_0$. We have

$$f_0\ r\ idcr \equiv_{(\mathsf{nxt}\ r)} f_1\ r\ idcr',$$

where $idcr'$ is obtained from $idcr$ by idc. We must show that

$$g\ (\mathsf{nxt}\ r)\ (f_0\ r\ idcr) \equiv_{(\mathsf{nxt}\ r)} g\ (\mathsf{nxt}\ r)\ (f_1\ r\ idcr').$$

But this follows by the extensionality of g.

The defining properties for a functor are easily verified.

[2] Remember that this means that the type is inhabited.

5. The coalgebra of computation trees

A possible first approach[3] to construct a final coalgebra representing the programs of Hancock/Setzer might be to work in the category of setoids [21,5]. The final coalgebra for the functor Prog ought to be defined by means of the set (List R) $\to$ C together with an appropriate equivalence relation. Given a morphism $g : B \to \mathsf{Prog}\ B$ the idea is now to define an element $tree_{g,b}$: (List R) $\to$ C for $b : B$ by

$$tree_{g,b}\ () := (g\ b)_{\mathsf{command}},$$
$$tree_{g,b}\ (l, r) := \begin{cases} (g\ b')_{\mathsf{command}} & \text{if } \mathsf{co}\, r \doteq tree_{g,b}\, l, \\ \text{some ``junk''} & \text{otherwise,} \end{cases}$$

where b' has to be defined simultaneously by means of $(g\ \_)_{\mathsf{next}_{\mathsf{El}}}$. However, this approach does not work. The reason is that we do not have $c \doteq c' \vee c \not\doteq c'$ for $c, c' : \mathsf{C}$ in general, i.e. identity on C must not be decidable. As a consequence, we cannot define $tree_{g,b}$ by case distinction as above. Instead, we have to prove our envisaged result by doing it the hard way[4]: we are going to define the object of the final coalgebra as a set of trees containing exactly the information a program needs to have. While well-founded trees can be represented in type theory by inductive types, for non-well-founded ones we have to resort to a more traditional mathematical definition: a tree is identified with the set of its paths. Specifically, for us paths are traces of computations represented by sequences of commands and responses (and states, but these are determined by the command). There must be a restriction, given by logical information in the sequence, that guarantees the agreement of states, commands and responses. This logical information ensures that the sequence is a possible computation path for the interface. It is given by identity proofs. In the next step we have to define subsets of traces corresponding to trees which then will give us the set of programs we are looking for. Subsets can be defined in type theory as predicates, that is, functions to Set. But to single out certain predicates which correspond to programs does not work because then the type of trees would not be a set. Alternatively, we can see subsets as functions to bool, but this is not possible in our case because we do not have decidability. The actual solution is intermediate: instead of using the whole universe of sets to characterize trees as subsets of paths, we define an ad hoc smaller and more manageable universe generated by the types of states, commands and responses and closed for identity and sigma types. Computation trees are then represented by functions on dependent sequences of commands and responses into this universe. We start by defining the set of possible computation paths of the interface:

Definition 4. Elements of $\mathsf{CTSeq}\ s$ for $s : \mathsf{S}$ are either of the form

$$(c, ids),$$

where $c : C$ and $ids : \mathsf{st}\ c \doteq s$, or of the form

$$(l,\ r,\ idc,\ c,\ ids),$$

where $l : \mathsf{CTSeq}\ s$, $r : \mathsf{R}$, $idc : \mathsf{co}\ r \doteq \mathsf{co\_}l$, $c : \mathsf{C}$, $ids : \mathsf{st}\ c \doteq \mathsf{nxt}\ r$ and $\mathsf{co\_}l$ denotes the last command of the sequence l, i.e.

$$\mathsf{co\_}(c, ids) = \mathsf{co\_}(l, r, idc, c, ids) = c.$$

Note that we have to define the function co mutually with the sets $\mathsf{CTSeq}\ s$, i.e. the definition is by induction–recursion [10]. The idea here is that a list represents an initial part of a possible program execution. The identities ensure that the list is accurate for the interface. We need some auxiliary notions:

Definition 5 (*Last state, predecessor*). We denote the last state of the sequence l : $\mathsf{CTSeq}\ s$ by $\mathsf{st\_}l$, i.e. $\mathsf{st\_}(c, ids) := s$, $\mathsf{st\_}(l, r, idc, c, ids) := \mathsf{nxt}\ r$. We denote the modified predecessor of the sequence l by $\mathsf{pd\_}l$, i.e. $\mathsf{pd\_}(c, ids) := (c, ids)$, $\mathsf{pd\_}(l, r, idc, c, ids) := l$.

[3] One of the referees of this paper suggested to explore this idea.

[4] "... the dwarfs found out how to turn lead into gold by doing it the hard way. The difference between that and the easy way is that the hard way works." Terry Pratchett, The Truth, 2000.

Definition 6 (*Append*). We define mutually

$$l_0 \star \langle r, idc\rangle \star l_1 : \mathsf{CTSeq}\ s$$

and an inhabitant of

$$\mathsf{co\_}l_1 \doteq \mathsf{co\_}(l_0 \star \langle r,\ idc\rangle \star l_1) \tag{1}$$

for $s : \mathsf{S}$, $l_0 : \mathsf{CTSeq}\ s$, $r : \mathsf{R}$, $idc : \mathsf{co}\ r \doteq \mathsf{co\_}l_0$, $l_1 : \mathsf{CTSeq}\ (\mathsf{nxt}\ r)$ by

$$l_0 \star \langle r, idc\rangle \star\ (c, ids) := (l_0, r, idc, c, ids),$$
$$l_0 \star \langle r, idc\rangle \star\ (l, r', idc', c, ids) := ((l_0 \star \langle r, idc\rangle \star l), r', idc'', c, ids),$$

where we obtain idc'' from idc' by the inhabitant of 1 which is defined as $\mathsf{refl}\ c$ in both cases.

Note that definition by cases is necessary in the definition of the inhabitant of 1 since otherwise the terms would not evaluate.

Proposition 7 (*Associativity of append*).

$$l_0 \star \langle r_0,\ idc_0\rangle \star (l_1 \star \langle r_1,\ idc_1\rangle \star l_2) \doteq (l_0 \star \langle r_0,\ idc_0\rangle \star l_1) \star \langle r_1,\ idc'_1\rangle \star l_2,$$

where idc'_1 is obtained from idc_1 by the inhabitant of

$$\mathsf{co\_}l_1 \doteq \mathsf{co\_}(l_0 \star \langle r_0,\ idc_0\rangle \star l_1)$$

due to 1.

Proof. Induction on l_2. If $l_2 \rightsquigarrow (c, ids)$ both sides of the equation evaluate to the same value. Let $l_2 \rightsquigarrow (l, r, idc, c, ids)$.
Let $c_1 := \mathsf{co\_}l_1$, $c'_1 := \mathsf{co\_}(l_0 \star \langle r_0,\ idc_0\rangle \star l_1)$.
Let $l_{left} := l_0 \star \langle r_0,\ idc_0\rangle \star (l_1 \star \langle r_1,\ idc_1\rangle \star l)$ and $l_{right} := (l_0 \star \langle r_0,\ idc_0\rangle \star l_1) \star \langle r_1,\ idc'_1\rangle \star l$. By I.H. we have

$$idl : l_{left} \doteq l_{right}.$$

Let $c_l := \mathsf{co\_}\ l$, $c'_l := \mathsf{co\_}(l_1 \star \langle r_1,\ idc_1\rangle \star l)$, $c_{left} := \mathsf{co\_}l_{left}$, $c_{right} := \mathsf{co\_}l_{right}$.
We have inhabitants of

$$c_l \doteq c'_l, \quad c'_l \doteq c_{left}, \quad c_l \doteq c_{right}$$

by which we obtain inhabitants

$$idc'_l : \mathsf{co}\ r \doteq c'_l, \quad idc_{left} : \mathsf{co}\ r \doteq c_{left}, \quad idc_{right} : \mathsf{co}\ r \doteq c_{right}$$

from idc. By idl we obtain a second inhabitant

$$idc'_{right} : \mathsf{co}\ r \doteq c_{right}$$

from idc_{left} and with $\mathsf{UIP}\ C$ we conclude that $idc'_{right} \doteq idc_{right}$ and

$$\langle l_{left},\ idc_{left}\rangle \doteq \langle l_{right},\ idc_{right}\rangle \tag{2}$$

by Principle 2. Now $l_0 \star \langle r_0,\ idc_0\rangle \star (l_1 \star \langle r_1\ idc_1\rangle \star l_2)$ evaluates to

$$(l_{left}, r, idc_{left}, c, ids)$$

and $(l_0 \star \langle r_0,\ idc_0\rangle \star l_1) \star \langle r_1,\ idc'_1\rangle \star l_2$ to

$$(l_{right}, r, idc_{right}, c, ids).$$

The claim follows by (2) with Principle 1. □

Remark. Note that to conclude that (2) holds, we have to prove that idc'_{right} equals idc_{right}. We obtained idc'_{right} from idc_{left} by shifting it along *idl*. Since we know nothing[5] about *idl* (we got *idl* from the I.H.) we know nothing about idc'_{right}. So to force the needed equality we apply UIP C.

Corollary 8.

$$(c, ids_0) \star \langle r, idc\rangle \star l \doteq (c, ids_1) \star \langle r, idc\rangle \star l$$

for $ids_0, ids_1 : (\mathsf{st}\ c) \doteq s$.

Proof. With UIP S. □

Corollary 9.

$$(c_0, ids_0) \star \langle r, idc\rangle \star l \doteq (c_1, ids_1) \star \langle r, idc'\rangle \star l$$

for $id : c_0 \doteq c_1, ids_i : (\mathsf{st}\ c_i) \doteq s\ (i = 0, 1)$ *and* $idc' = \mathsf{subst}\ id\ idc$.

Proof. Case $id = \mathsf{refl}\ c$. □

We are going to define a universe U. The definition is by induction–recursion [10]. The universe U is a relatively small universe. It contains names for the sets S, C, R and is closed only under the identity and sigma type formers. For the general rôle of universes in type theory and the proof theoretic strength gained by (much larger) universes compare [36,41].

Definition 10 (*Universe*). We define mutually

```
U : Set
= data NameS | NameC | NameR |
       NameId (u : U)(e₁, e₂ : set u) |
       NameSig (u : U)(f : (e : set u) → U)
```

and

```
set(u : U) : Set
by
 set NameS           = S
 set NameC           = C
 set NameR           = R
 set (NameId u e₁ e₂) = (e₁ ≐_(set u) e₂)
 set (NameSig u f)   = ∑(e : (set u).(set (f e))).
```

We write NIdC for NameId NameC.

We want to define computation trees as functions $T : \mathsf{CTSeq}\ s \to \mathsf{U}$ with the following properties:
(1) There is exactly one root $c : \mathsf{C}$ for the tree.
(2) For every $l : \mathsf{CTSeq}\ s$ which is a node of the tree and for every $r : \mathsf{R}$ *suitable* for l there is exactly one successor, i.e. one c such that (l, r, idc, c, ids) is a node of the tree.
(3) For every $l : \mathsf{CTSeq}\ s$ which is a node of the tree the predecessor of l is a node of the tree too.
Where a list l is a node of the tree if set $(T\ \ l)$ is inhabited and $r : \mathsf{R}$ is suitable for l if $\mathsf{co}\,r \doteq \mathsf{co}\_l$. Technically a computation tree will be a dependent tuple of a function T together with a witness that the function fulfils the

[5] At least we do not know if types depending on *idl* are inhabited.

properties above (Definition 12). The properties are expressed by sigma types (Definition 11). We formalize this ideas as follows:

Definition 11. For $s : \mathsf{S}$, $T : \mathsf{CTSeq}\ s \to \mathsf{U}$ let $\Phi_1\ s\ T$ be

$$\begin{array}{lll} \mathsf{sig} & \mathsf{root} & : \mathsf{C} \\ & \mathsf{id}^{\mathsf{S}}_{\mathsf{ro}} & : \mathsf{st}\ \ \mathsf{root} \doteq s \\ & \mathsf{root}_{\mathsf{ex}} & : \mathsf{set}\ (\mathsf{T}(\mathsf{root}, \mathsf{id}^{\mathsf{S}}_{\mathsf{ro}})) \\ & \mathsf{root}_{\mathsf{uni}} & : \ \forall c : \mathsf{C}, idsc : \mathsf{st}\ c \doteq s.\ \mathsf{set}\ (T\ (c,\ idsc)) \Rightarrow c \doteq \mathsf{root}. \end{array}$$

For $s : \mathsf{S}$, $T : \mathsf{CTSeq}\ s \to \mathsf{U}$, $l : \mathsf{CTSeq}\ s$, $e : \mathsf{set}\ (T\ l)$, $r : \mathsf{R}$ and $idcr : \mathsf{co}\ r \doteq (\mathsf{co}\_l)$ let $\Phi_2\ s\ T\ l\ e\ r\ idcr$ be

$$\begin{array}{lll} \mathsf{sig} & \mathsf{command} & : \mathsf{C} \\ & \mathsf{id}^{\mathsf{S}}_{\mathsf{co}} & : \mathsf{st}\ \ \mathsf{command} \doteq \mathsf{nxt}\ r \\ & \mathsf{command}_{\mathsf{ex}} & : \mathsf{set}\ (T(l, r, idcr, \mathsf{command}, \mathsf{id}^{\mathsf{S}}_{\mathsf{co}})) \\ & \mathsf{command}_{\mathsf{uni}} & : \ \forall c : \mathsf{C}, idsc : \mathsf{st}\ c \doteq \mathsf{nxt}\ r.\ \mathsf{set}\ (T\ \ (l, r, idcr, c, idsc)) \Rightarrow \\ & & \qquad\qquad\qquad\qquad\qquad\qquad c \doteq \mathsf{command}\,. \end{array}$$

For $s : \mathsf{S}$, $T : \mathsf{CTSeq}\ s \to \mathsf{U}$, $l : \mathsf{CTSeq}\ s$ and $e : \mathsf{set}\ (T\ l)$ let $\Phi_3\ s\ T\ l\ e$ be

$$\mathsf{set}\ (T\ \ (\mathsf{pd}\_l)).$$

Let $\Phi\ s\ T$ be $(\Phi_1\ s\ T) \wedge (\Phi_2\ s\ T) \wedge (\Phi_3\ s\ T)$.

Note the natural way in which we make use of dependent types in this definition: we quantify in Φ only over those l which are nodes of the tree T: argument $e : \mathsf{set}\ (T\ l)$ in Φ_1 and Φ_2. This will play an important role later. We are now able to define the family of sets in the object part of the final coalgebra:

Definition 12 (*Computation trees*).

$$\begin{array}{ll} \multicolumn{2}{l}{\mathsf{CT}\ (s : \mathsf{S}) : \mathsf{Set}} \\ = & \mathsf{sig}\ \mathsf{tree} : \mathsf{CTSeq}\ s \to \mathsf{U} \\ & \qquad \mathsf{phi} : \Phi\ s\ \mathsf{tree}. \end{array}$$

Before we define an equivalence relation on this family we declare the morphism of the final coalgebra. We single out the command of each tree by using the witness for the property Φ_1.

Definition 13 (*Command of a computation tree*). For $ct : \mathsf{CT}\ s$: with $ct_{\mathsf{phi}} \rightsquigarrow (\varphi_1,\ \varphi_2,\ \varphi_3)$

$$\begin{array}{l} \mathsf{co}\_ct := \varphi_{1_{\mathsf{root}}}, \\ \mathsf{id}^{\mathsf{S}}_{\mathsf{co}}\_ct := \varphi_{1_{\mathsf{id}^{\mathsf{S}}_{\mathsf{ro}}}}. \end{array}$$

The program that we obtain after doing one computation step and receiving a response r is represented by the subtree at branch r. A subtree is given by taking the tree function on another position. The argument is constructed by means of the append function on lists.

Definition 14 ($\mathsf{elim}_{\mathsf{tree}}$). For $ct : \mathsf{CT}\ s$, $r : \mathsf{R}$ and $idc : \mathsf{co}\ r \doteq \mathsf{co}\_ct$ let

$$\mathsf{elim}_{\mathsf{tree}}\ s\ ct\ r\ idc\ \ :\ \ \mathsf{CTSeq}\ (\mathsf{nxt}\ r) \to \mathsf{U}$$

given by

$$\lambda l : \mathsf{CTSeq}\ (\mathsf{nxt}\ r)\ .\ (ct_{\mathsf{tree}}\ ((c,\ ids) \star (r,\ idc) \star l)),$$

where $c = \mathsf{co}\_ct$ and $ids = \mathsf{id}^{\mathsf{S}}_{\mathsf{co}}\_ct$.

We need to prove that the defined function has the properties Φ_1–Φ_3.

Proposition 15. *For* $ct : \mathsf{CT}\ s, r : \mathsf{R}$ *and* $idc : (\mathsf{co}\ r) \doteq (\mathsf{co}\_ct)$

$$\Phi_1\ (\mathsf{nxt}\ r)\ (\mathsf{elim}_{\mathsf{tree}}\ s\ ct\ r\ idc).$$

Proof. Let $c = \mathsf{co}\_ct$ and $ids = \mathsf{id}^{\mathsf{S}}_{\mathsf{co}}\_ct$. The inhabitant of $\Phi_1\ s\ ct_{\mathsf{tree}}$ gives an inhabitant $e : \mathsf{set}(ct_{\mathsf{tree}}\ s\ (c,\ ids))$. The inhabitant of $\Phi_2\ s\ ct_{\mathsf{tree}}\ (c,\ ids)\ e\ r\ idc$ proves the claim. □

Proposition 16. *For* $ct : \mathsf{CT}\ s, r : \mathsf{R}$ *and* $idc : (\mathsf{co}\ r) \doteq (\mathsf{co}\_ct)$

$$\Phi_2\ (\mathsf{nxt}\ r)\ (\mathsf{elim}_{\mathsf{tree}}\ s\ ct\ r\ idc).$$

Proof. Let $l : \mathsf{CTSeq}\ (\mathsf{nxt}\ r)$, $e : \mathsf{set}\ (\mathsf{elim}_{\mathsf{tree}}\ s\ ct\ r\ idc\ l)$, $r' : \mathsf{R}$, $idc' : (\mathsf{co}\ r') \doteq (\mathsf{co}\_l)$. Let $c = \mathsf{co}\_ct$ and $ids = \mathsf{id}^{\mathsf{S}}_{\mathsf{co}}\_ct$. By (1) we get an inhabitant $idc'' : (\mathsf{co}\ r') \doteq (\mathsf{co}\_((c,\ ids) \star \langle r,\ idc\rangle \star l)$ from idc' and the inhabitant of $\Phi_2\ s\ ct_{\mathsf{tree}}\ ((c,\ ids) \star \langle r,\ idc\rangle \star l)\ e\ r'\ idc''$ proves the claim. □

Proposition 17. *For* $ct : \mathsf{CT}\ s, r : \mathsf{R}$ *and* $idc : (\mathsf{co}\ r) \doteq (\mathsf{co}\_ct)$

$$\Phi_3\ (\mathsf{nxt}\ r)\ (\mathsf{elim}_{\mathsf{tree}}\ s\ ct\ r\ idc).$$

Proof. Induction on $l : \mathsf{CTSeq}\ (\mathsf{nxt}\ r)$. □

The morphism part of the final coalgebra is now given by:

Definition 18 (*elim*). For $ct : \mathsf{CT}\ s$ we define

$$\mathsf{elim}\ s\ ct\ \ :\ \ \mathsf{Prog}\ s\ \mathsf{CT}$$

by

$$\langle \mathsf{co}\_ct,\ \mathsf{id}^{\mathsf{S}}_{\mathsf{co}}\_ct,\ next_{El}\rangle,$$

where $next_{El}\ r\ idc : \mathsf{CT}\ (\mathsf{nxt}\ r)$ is given by $\mathsf{elim}_{\mathsf{tree}}\ s\ ct\ r\ idc$ and Propositions 15–17 for $r : \mathsf{R}$ and $idc : (\mathsf{co}\ r) \doteq (\mathsf{co}\_ct)$.

We write $\mathsf{next}_{\mathsf{El}}\_ct$ for $(\mathsf{elim}\_ct)_{\mathsf{next}_{\mathsf{El}}}$.

5.1. Bisimulation

We still need to define an equivalence relation on CT. The equivalence relation we are going to define expresses when two programs behave in the same way. This is the case if they start with equal commands and give equivalent programs for equal responses. The function elim gives a labelled transition system. There is a transition $r : \mathsf{R}$ between trees T_0 and T_1 if T_1 is the subtree of T_0 at branch r. Since this transition system is image finite we can define bisimulation by means of natural induction.

Definition 19 (*Bisimulation*). For $ct, ct' : \mathsf{CT}\ s, n : \mathsf{N}$ we define

$$ct \sim_n ct' : \mathsf{Set}$$

by

$$\begin{aligned} ct \sim_{\mathsf{zero}} ct' &= \mathsf{True}, \\ ct \sim_{\mathsf{succ}\ n} ct' &= \mathsf{sig}\quad \mathsf{idc} : c \doteq c' \\ &\qquad\quad \mathsf{fct} : (r : \mathsf{R}, idcr : (\mathsf{co}\ r) \doteq c) \to f\ r\ idcr \sim_n f'\ r\ idcr' \end{aligned}$$

and

$$\begin{aligned} ct \sim ct' &: \mathsf{Set} \\ &= \forall n : \mathsf{N}.\ ct \sim_n ct', \end{aligned}$$

where $\mathsf{elim}\ s\ ct \rightsquigarrow \langle c, idc, f\rangle$, $\mathsf{elim}\ s\ ct' \rightsquigarrow \langle c', idc', f'\rangle$ and $idcr'$ is obtained from $idcr$ by idc.

Proposition 20. $\sim$ *is an equivalence relation on* CT.

Proof. Straightforward. □

Proposition 21.

$$ct \sim ct' \ \Leftrightarrow\ \mathsf{elim}\ s\ ct \sim_{\mathsf{Prog}} \mathsf{elim}\ s\ ct'$$

for $ct, ct' : \mathsf{CT}\ s$.

Proof. "⇒" Follows with UIP C.
"⇐" Trivial. □

Corollary 22. $\mathsf{elim} : \mathsf{CT} \to \mathsf{Prog}\ \mathsf{CT}$ *is extensional.*

This means that elim is a coalgebra morphism. We are going to prove that $\mathsf{elim} : \mathsf{CT} \to \mathsf{Prog}\ \mathsf{CT}$ is a final coalgebra for Prog.

6. The unique morphism T into the final coalgebra

Let $B : \mathsf{S} \to \mathsf{Set}$ and $g : (s : \mathsf{S}, B\ s) \to \mathsf{Prog}\ B\ s$. We keep B, g fixed for the rest of the article. We write $\mathsf{co\_}b$, $\mathsf{id}^{\mathsf{S}}_{\mathsf{co}}\mathsf{\_}b$ and $\mathsf{next}_{\mathsf{El}}\ s\ b$ for $(g\ s\ b)_{\mathsf{command}}$, $(g\ s\ b)_{\mathsf{id}^{\mathsf{S}}_{\mathsf{co}}}$, $(g\ s\ b)_{\mathsf{next}_{\mathsf{El}}}$, respectively, where $b : B\ s$. We must find a unique morphism $\mathsf{T} : B \to \mathsf{CT}$ with $\mathsf{elim} \circ \mathsf{T} = \mathsf{Prog}\ \mathsf{T} \circ g$, i.e. $\mathsf{elim}\ s\ (\mathsf{T}\ s\ b) \sim_{\mathsf{Prog}} (\mathsf{Prog}\ \mathsf{T})\ s\ (g\ s\ b)$ for $s : \mathsf{S}$, $b : B\ s$. We get T by defining mutually the function value $\mathsf{T}_{\mathsf{tree}}\ s\ b\ l : \mathsf{U}$ for $l : \mathsf{CTSeq}\ s$ and an element of $B\ (\mathsf{nxt}\ r)$ for those l which are nodes of $\mathsf{T}_{\mathsf{tree}}\ s\ b$ where r is a response of $\mathsf{co\_}l$. This element is essentially the element which we get if we follow g along the responses which occur in l including r. The list (c, ids) is a node of the tree $\mathsf{T}_{\mathsf{tree}}\ s\ b$ if c is the command played by g at b, i.e. $(\mathsf{co\_}b) \doteq c$. The list $(l, r, \_, c, \_)$ is a node of the tree $\mathsf{T}_{\mathsf{tree}}\ s\ b$ if l is a node of $\mathsf{T}_{\mathsf{tree}}\ s\ b$ and c is the command played by g at the element of $B\ (\mathsf{nxt}\ r)$ described above. Things again become quite involved since we have to shift the identities to meet the typing requirements.

Definition 23. We define mutually

$$\mathsf{T}_{\mathsf{tree}}\ s\ b\ l : \mathsf{U}$$

and

$$\mathsf{A}\ s\ b\ l\ r\ idc\ e : B\ (\mathsf{nxt}\ r)$$

for $s : \mathsf{S}$, $b : B\ s$, $l : \mathsf{CTSeq}\ s$, $r : \mathsf{R}$, $idc : \mathsf{co}\ r \doteq \mathsf{co\_}l$ and $e : \mathsf{set}\ (\mathsf{T}_{\mathsf{tree}}\ s\ b\ l)$ by

$$\begin{aligned} \mathsf{T}_{\mathsf{tree}}\ s\ b\ (c, ids) &:= \mathsf{NIdC}\ c\ (\mathsf{co\_}b), \\ \mathsf{T}_{\mathsf{tree}}\ s\ b\ (l', r', idc', c, ids) &:= \mathsf{NameSig}\ (\mathsf{T}_{\mathsf{tree}}\ s\ b\ l') \\ &\qquad\qquad (\lambda e : \mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s\ b\ l').\ \mathsf{NIdC}\ c\ (c'\ e)), \end{aligned}$$

where $c'\ e := \mathsf{co\_}(\mathsf{A}\ s\ b\ l'\ r'\ idc'\ e)$ and

$$\begin{aligned} \mathsf{A}\ s\ b\ (c, ids)\ r\ idc\ e &:= \mathsf{next}_{\mathsf{El}}\ s\ b\ r\ idce, \\ \mathsf{A}\ s\ b\ (l', r', idc', c, ids)\ r\ idc\ e &:= \mathsf{next}_{\mathsf{El}}\ (\mathsf{nxt}\ r')\ b'\ r\ idc'', \end{aligned}$$

where in the first case *idce* is the composition of *idc* and *e*, and in the second case $b' := \mathsf{A}\ s\ b\ l'\ r'\ idc'\ e_{\mathsf{fst}}$, $e_{\mathsf{snd}} : c \doteq (\mathsf{co}\_b')$ and $idc'' := \mathsf{subst}\ e_{\mathsf{snd}}\ idc$.

We lift UIP C to $\mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s\ b\ l)$:

Proposition 24.

$$\forall p, q : \mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s\ b\ l) \Rightarrow p \doteq q$$

for $s : \mathsf{S}, b : B\ s, l : \mathsf{CTSeq}\ s$.

Proof. If $l \rightsquigarrow (c, ids)$ this is UIP C.

Let $l \rightsquigarrow (l', r, idc, c, ids), p, q : \mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s\ b\ l)$ with $p \rightsquigarrow \langle p', idcp\rangle$ and $q \rightsquigarrow \langle q', idcq\rangle$. We have $id : p' \doteq_{\mathsf{set}(\mathsf{T}\ s\ b\ l')} q'$ by I.H. and $idcp' \doteq idcq$ by UIP C where we obtain $idcp'$ from $idcp$ by id. This proves the claim. □

Corollary 25.

$$\mathsf{A}\ s\ b\ l\ r\ idc\ p \doteq \mathsf{A}\ s\ b\ l\ r\ idc\ q$$

for $s : \mathsf{S}, b : B\ s, l : \mathsf{CTSeq}\ s, r : \mathsf{R}, idc : (\mathsf{co}\ r) \doteq (\mathsf{co}\_l)$ *and* $p, q : \mathsf{set}\ (\mathsf{T}_{\mathsf{tree}}\ s\ b\ l)$.

Proof. Immediate from Proposition 24. □

The following three propositions state that $\mathsf{T}_{\mathsf{tree}}\ s\ b$ is indeed an element of CT, i.e. that $\mathsf{T}_{\mathsf{tree}}\ s\ b$ fulfils the properties Φ_1–Φ_3.

Proposition 26. *For* $s : \mathsf{S}, b : B\ s$

$$\Phi_1\ s\ (\mathsf{T}_{\mathsf{tree}}\ s\ b).$$

Proof. The following term proves the claim:

$$\langle \mathsf{co}\_b, \mathsf{id}^{\mathsf{S}}_{\mathsf{co}}\_b, \mathsf{refl}\ (\mathsf{co}\_b), \mathrm{root}_{\mathrm{uni}_\mathrm{T}}\rangle,$$

where $\mathrm{root}_{\mathrm{uni}_\mathrm{T}}\ c\ idsc\ p := p$ for $c : \mathsf{C}, idsc : (\mathsf{st}\ c) \doteq s$ and $p : \mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s\ b\ (c, idsc))$. □

Proposition 27. *For* $s : \mathsf{S}, b : B\ s$

$$\Phi_2\ s\ (\mathsf{T}_{\mathsf{tree}}\ s\ b).$$

Proof. Let $l : \mathsf{CTSeq}\ s, p : \mathsf{set}\ (\mathsf{T}_{\mathsf{tree}}\ s\ b\ l), r : \mathsf{R}, idcr : (\mathsf{co}\ r) \doteq (\mathsf{co}\_l)$.

For $x : \mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s\ \ b\ l)$ let $b'\ x := \mathsf{A}\ s\ b\ l\ r\ idcr\ x$, $c'\ x := \mathsf{co}\_(b'\ x)$ and $ids'\ x := \mathsf{id}^{\mathsf{S}}_{\mathsf{co}}\_(b'\ x)$. The following term proves the claim:

$$\langle c'\ p, ids'\ p, (p, \mathsf{refl}\ c'\ p), \mathrm{command}_{\mathrm{uni}_\mathrm{T}}\rangle,$$

where $\mathrm{command}_{\mathrm{uni}_\mathrm{T}}\ c\ idsc\ q := \mathsf{subst}\ id\ q_{\mathsf{snd}}$ and $id : q_{\mathsf{fst}} \doteq p$ the inhabitant according to Proposition 24 for $c : \mathsf{C}, idsc : \mathsf{st}\, c \doteq \mathsf{nxt}\, r$ and $q : \mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s\ b\ (l \star \langle r, idcr\rangle \star (c, idsc)))$. □

Proposition 28. *For* $s : \mathsf{S}, b : B\ s$

$$\Phi_3\ s\ (\mathsf{T}_{\mathsf{tree}}\ s\ b).$$

Proof. Obvious. □

Definition 29. Let $\mathsf{T}(s : \mathsf{S})(b : B\ s) : \mathsf{CT}\ s$ be

$$\langle \mathsf{T}_{\mathsf{tree}}\ s\ b, \mathsf{T}_{\mathsf{phi}}\ s\ b\rangle,$$

where $\mathsf{T}_{\mathsf{phi}}\ s\ b$ is given by Propositions 26–28.

We postpone the proof that T is extensional.

7. The repetition of the unique morphism T

We want to prove that T is the unique morphism making the coalgebra square below commute.

$$\begin{tikzcd} B \arrow[r, "g"] \arrow[d, dashed, "\mathsf{T}"'] & \mathsf{Prog}\ B \arrow[d, "\mathsf{Prog}\ \mathsf{T}"] \\ \mathsf{CT} \arrow[r, "\mathsf{elim}"'] & \mathsf{Prog}\ \mathsf{CT} \end{tikzcd}$$

That means we have to prove

$$b_0 \equiv b_1 \Rightarrow (\mathsf{Prog}\ \mathsf{T} \circ g)\_b_0 \sim (\mathsf{elim} \circ \mathsf{T})\_b_1$$

for $s : \mathsf{S}$ and $b_0, b_1 : B\ s$, where $\equiv$ denotes the equivalence relation on B. We have

$$\begin{aligned} ct_0 \sim_n ct_1 \;&\Leftrightarrow\; \mathsf{co}\_ct_0 \doteq \mathsf{co}\_ct_1 \text{ and} \\ &\quad\;\; \mathsf{next}_{\mathsf{El}}\_ct_0\ r_0\ idcr_0 \sim_{n-1} \mathsf{next}_{\mathsf{El}}\_ct_1\ r_0\ idcr'_0 \\ &\Leftrightarrow\; \ldots \text{ and} \\ &\quad\;\; \mathsf{next}_{\mathsf{El}}\_(\mathsf{next}_{\mathsf{El}}\_ct_0\ r_0\ idcr_0)\ r_1\ idcr_1 \sim_{n-2} \\ &\quad\;\;\;\; \mathsf{next}_{\mathsf{El}}\_(\mathsf{next}_{\mathsf{El}}\_ct_1\ r_0\ idcr'_0)\ r_1\ idcr'_1 \\ &\Leftrightarrow\; \ldots \end{aligned}$$

for $ct_0, ct_1 : \mathsf{CT}\ s$. This observation leads to the definition of the repetition $\mathsf{T}_{\mathsf{Rep}}$ of T which we use in the following to prove the coalgebra property. We define the repetition $\mathsf{T}_{\mathsf{Rep}}$ of T for every sequence $l : \mathsf{CTSeq}\ s$ which belongs to $\mathsf{T}_{\mathsf{tree}}\ s\ b$. Essentially this will be the subtree of $\mathsf{T}_{\mathsf{tree}}$ which we get when we follow the tree along the path l. We want to define this by recursion on l. Again this cannot be done in a straightforward way, since the elements we get by the induction hypothesis do not have the desired type. That means we have to shift them along certain identities which must be defined simultaneously. Therefore we define mutually

$$\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l\ p : \mathsf{CT}(\mathsf{st}\_l)$$

and identities

$$\mathsf{co}\_l \doteq_{\mathsf{C}} \mathsf{co}\_(\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l\ p), \tag{3}$$

$$(\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l\ p)_{\mathsf{tree}}\ l' \doteq_{\mathsf{U}} \mathsf{T}_{\mathsf{tree}}\ s\ b\ (l\#l'), \tag{4}$$

where $s : \mathsf{S}$, $b : B\ s$, $l : \mathsf{CTSeq}\ s$, $p : \mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s\ b\ l)$, $l' : \mathsf{CTSeq}(\mathsf{st}\_l)$ and

$$\begin{aligned} (c, ids)\#l' &:= l', \\ (l_0, r, idc, c, ids)\#l' &:= (l_0 \star \langle r, idc\rangle \star l'). \end{aligned}$$

The second identity (in U) is needed to prove the first one (in C). Note that $l\#(c, ids) \rightsquigarrow l$ if $l \rightsquigarrow (c, ids)$ or $l \rightsquigarrow (l_0, r, idc, c, ids)$.

Definition 30 (*Repetition of* T). We define mutually
$\mathsf{T}_{\mathsf{Rep}}(s : \mathsf{S})(b : B\ s)(l : \mathsf{CTSeq}\ s)(p : \mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s\ b\ l)) : \mathsf{CT}\ (\mathsf{st\_}l)$
by
$\mathsf{T}_{\mathsf{Rep}}\ s\ b\ (c, ids)\ p \qquad = \mathsf{T}\ s\ b$
$\mathsf{T}_{\mathsf{Rep}}\ s\ b\ (l'\ r\ idc\ c\ ids)\ p = \mathsf{next}_{\mathsf{El}}\mathsf{\_}(\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l'\ p')\ r\ idc',$
where the witness saying that l' belongs to the tree $\mathsf{T}\,s\,b$ is given by $p' = \varphi_3\ s\ b\ l\ p$ where $\varphi_3 : \Phi_3\ s\ (\mathsf{T}_{\mathsf{tree}}\ s\ b)$ as in Proposition 28. The identity idc' is obtained from idc by identity 3. Identity 3 is defined in the next step.

We define an inhabitant of

$$\mathsf{co\_}l \doteq \mathsf{co\_}(\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l\ p)$$

by

$$\begin{array}{ll} p & \text{if } l = (c, ids), \\ \varphi_{1_{\mathsf{root}_{\mathsf{uni}}}}\ c\ ids\ p' & \text{if } l = (l', r, idc, c, ids), \end{array}$$

where $\varphi_1 : \Phi_1\mathsf{\_}(\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l\ p)$ as given by Proposition 26. The witness p' saying that (c, ids) belongs to the tree $\mathsf{T}_{\mathsf{Rep}}\,s\,b\,l\,p$ is obtained from p by identity (4) since $l\#(c, ids) \rightsquigarrow l$. Identity (4) is defined in the next step.

To complete the definition we must define an inhabitant of

$$(\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l\ p)_{\mathsf{tree}}\ l' \doteq \mathsf{T}_{\mathsf{tree}}\ s\ b\ (l\#l')$$

for $s : \mathsf{S}$, $b : B\ s$, $l : \mathsf{CTSeq}\ s$, $p : \mathsf{set}(\mathsf{T}\ s\ b\ l)$ and $l' : \mathsf{CTSeq}\ (\mathsf{st\_}l)$.

In the case $l \rightsquigarrow (c, ids)$ an inhabitant of this type is given by

$$\mathsf{refl}\ (\mathsf{T}_{\mathsf{tree}}\ s\ b\ (l\#l'))$$

since both sides of the equation evaluate to the same value.

For $l \rightsquigarrow (l_0, r, idc, c, ids)$ let p' be as above,

$$\begin{aligned} s_0 &:= \mathsf{st\_}l_0, \\ c_{l_0} &:= \mathsf{co\_}(\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l_0\ p'), \\ idcs_{l_0} &:= \mathsf{id}^{\mathsf{S}}_{\mathsf{co\_}}(\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l_0\ p'), \\ sl_0 &:= (c_{l_0}, idcs_{l_0}), \\ sl_1 &:= (c_0, idcs_0), \end{aligned}$$

where $l_0 = (\ldots, c_0, idcs_0)$ and idc' is obtained from idc by identity (3). Let

$$\begin{aligned} left &= (\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l_0\ p')_{\mathsf{tree}}\ (sl_0 \star \langle r, idc'\rangle \star l'), \\ middle &= \mathsf{T}_{\mathsf{tree}}\ s\ b\ (l_0\#(sl_0 \star \langle r, idc'\rangle \star l')), \\ right &= \mathsf{T}_{\mathsf{tree}}\ s\ b\ (l_0 \star \langle r, idc\rangle \star l'). \end{aligned}$$

We must prove $left \doteq right$. We have $sl_0 \star \langle r, idc'\rangle \star l' \doteq sl_1 \star \langle r, idc\rangle \star l'$ by Corollary 9 and by I.H.

$$left \doteq middle.$$

If $l_0 \rightsquigarrow (c_0, idcs_0)$ then

$$middle \doteq right$$

by Principle 1 and we are done.

If $l_0 \rightsquigarrow (l_1, r_0, idcr_0, c_0, idcs_0)$ then

$$\begin{aligned} l_1 \star \langle r_0, idcr_0\rangle \star (sl_0 \star \langle r, idc\rangle \star l') &\doteq l_1 \star \langle r_0, idcr_0\rangle \star (sl_1 \star \langle r, idc'\rangle \star l') \\ &\doteq (l_1 \star \langle r_0, idcr_0\rangle \star sl_1) \star \langle r, idc'\rangle \star l' \\ &\doteq l_0 \star \langle r, idc'\rangle \star l', \end{aligned}$$

where the first equation follows by Principle 1 and the second by the associativity of $\star$. Principle 1 gives

$$middle \doteq right$$

and we are done again. This finishes the definition of $\mathsf{T}_{\mathsf{Rep}}$.

Remark. Note that we could define the repetition of *ct* for arbitrary $ct : \mathsf{CT}\ s$. Therefore we can proceed in a similar way as above. We just need to replace p by $\varphi_{1_{\mathsf{root}_{\mathsf{uni}}}}\ c\ ids\ p$ in the first case of the construction of the inhabitant of identity (3) where φ_1 is the witness for $\Phi_1\ s\ ct$. However, since this greater generality has no particular advantage for us we work with the definition above.

As a corollary to Proposition 24 we get:

Corollary 31.

$$\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l\ p \doteq \mathsf{T}_{\mathsf{Rep}}\ s\ b\ l\ q$$

for $p, q : \mathsf{T}_{\mathsf{tree}}\ s\ b\ l$.

We need some auxiliary definitions. Let

$$\begin{aligned}\mathsf{nxtS}\ s\ (c, ids)\ r &:= \mathsf{nxt}\ r,\\ \mathsf{nxtS}\ s\ (l', r', idc, c, ids)\ r &:= \mathsf{nxtS}\ s\ l'\ r',\end{aligned}$$

$$\begin{aligned}\mathsf{pred}\ s\ (c', ids')\ r\ idc\ c\ ids &:= (c, ids),\\ \mathsf{pred}\ s\ (l', r', idc', c', ids')\ r\ idc\ c\ ids &:= ((\mathsf{pred}\ s\ l'\ r'\ idc'\ c'\ ids'), r, idcr, c, ids),\end{aligned}$$

where *idcr* is obtained from *idc* by the simultaneously defined inhabitant of

$$c \doteq \mathsf{co}\_(\mathsf{pred}\ s\ l\ r\ idc\ c\ ids), \tag{5}$$

which is given in both cases by $\mathsf{refl}\ c$. Note that definition by cases is necessary again to define this inhabitant. The operation $\mathsf{pred}\ \_\ l\_$ cuts off the first command and response of the list l. Since this is only possible for lists of the form (l', r', idc', c', ids') we use the auxiliary arguments r, *idc*, c and *ids*. The obtained list is an inhabitant of $\mathsf{CTSeq}\ (\mathsf{nxtS}\ s\ l\ r)$. Further, we define an inhabitant of $B\ (\mathsf{nxtS}\ s\ l\ r)$ by

$$\begin{aligned}\mathsf{next_B}\ s\ (c', ids')\ r\ idc\ c\ ids\ b\ (id_0, id_1) &:= \mathsf{A}\ s\ b\ (c', ids')\ r\ idc\ id_0,\\ \mathsf{next_B}\ s\ (l', r', idc', c', ids')\ r\ idc\ c\ ids\ b\ (p', id_1) &:= \mathsf{next_B}\ s\ l'\ r'\ idc'\ c'\ ids'\ b\ p',\end{aligned}$$

where $b : B\ s$ and $p : \mathsf{set}\ (\mathsf{T}_{\mathsf{tree}}\ s\ b\ (l, r, idc, c, ids))$ for $p \rightsquigarrow (id_0, id_1)$, $p \rightsquigarrow (p', id_1)$ and $l \rightsquigarrow (c', ids')$, $l \rightsquigarrow (l', r', idc', c', ids')$, respectively. The inhabitant $\mathsf{next_B}\ \_\ l\ r\ \_$ is calculated by doing essentially only the first step in the calculation of $\mathsf{A}\ \_\ l\ r\ \_$. Whereas $\mathsf{A}\ \_\ l\ r\ \_$ gives us an element of $B\ (\mathsf{nxt}\ r)$ by following all responses in l including r, $\mathsf{next_B}\ \_\ l\ r\ \_$ is doing only the first step. The following proposition states that we get equal elements in $B\ (\mathsf{nxt}\ r_0)$ whether we apply A on b and a sequence $(l, r, idcr, c, idsc)$ or do one step from b along this sequence and apply A on the new b_{n} and the sequence obtained from $(l, r, idcr, c, idsc)$ by pred above.

Proposition 32. *For* $s : \mathsf{S}, l : \mathsf{CTSeq}\ s, r, r_0 : \mathsf{R}, c : \mathsf{C}, b : B\ s$,

$$\begin{aligned}idcr &: \mathsf{co}\ r \doteq \mathsf{co}\_l,\\ idsc &: \mathsf{st}\ c \doteq \mathsf{nxt}\ r,\\ idcr_0 &: \mathsf{co}\ r_0 \doteq c,\end{aligned}$$

$p : \mathsf{set}\ (\mathsf{T}_{\mathsf{tree}}\ s\ b\ (l, r, idcr, c, idsc))$, $q : \mathsf{set}\ (\mathsf{T}_{\mathsf{tree}}\ s_{\mathsf{n}}\ b_{\mathsf{n}}\ l_{\mathsf{p}})$, *where*

$$\begin{aligned}s_{\mathsf{n}} &= \mathsf{nxtS}\ s\ l\ r,\\ b_{\mathsf{n}} &= \mathsf{next_B}\ s\ l\ r\ idcr\ c\ idsc\ b\ p,\\ l_{\mathsf{p}} &= \mathsf{pred}\ s\ l\ r\ idcr\ c\ idsc,\end{aligned}$$

we have

$$\mathsf{A}\ s\ b\ (l, r, idcr, c, idsc)\ r_0\ idcr_0\ p \doteq \mathsf{A}\ s_{\mathsf{n}}\ b_{\mathsf{n}}\ l_{\mathsf{p}}\ r_0\ idcr'_0\ q,$$

where $idcr'_0$ is obtained from $idcr_0$ by identity (5).

Proof. Case $l \rightsquigarrow (c', ids)$, $p \rightsquigarrow (id_0, id_1)$.

Then $idcr'_0$ evaluates to $idcr_0$. Let $idcr_1$, $idcr_2$ be obtained from $idcr_0$ by id_1, q, respectively. By UIP C we have

$$idcr_1 \doteq idcr_2$$

and by Principle 1 we get

$$\begin{aligned}\mathsf{A}\ s\ b\ ((c', ids), r, idcr, c, idsc)\ r_0\ idcr_0\ (id_0, id_1) &\doteq f\ idcr_1\\ &\doteq f\ idcr_2\\ &\doteq \mathsf{A}\ s_{\mathsf{n}}\ b_{\mathsf{n}}\ l_{\mathsf{p}}\ r_0\ idcr_0\ q,\end{aligned}$$

where $f = (g\ (\mathsf{nxt}\ r)\ ((g\ s\ b)_{\mathsf{next_{El}}}\ r\ idcr'))_{\mathsf{next_{El}}}\ r_0$ and $idcr'$ is obtained from $idcr$ by id_0.

Case $l \rightsquigarrow (l', r', idc', c', ids)$, $p \rightsquigarrow (p', id_1)$, $q \rightsquigarrow (q', id_2)$.

Then again $idcr'_0$ evaluates to $idcr_0$. We define

$$f : (x : B\ (\mathsf{nxt}\ r)) \to IdC\ x \to B\ (\mathsf{nxt}\ r_0),$$

where $IdC\ x := ((\mathsf{co}\ r_0) \doteq (\mathsf{co\_}x))$ by

$$f\ x\ y = (g\ (\mathsf{nxt}\ r)\ x)_{\mathsf{next_{El}}}\ r_0\ y$$

for $x : B\ (\mathsf{nxt}\ r)$, $y : IdC\ x$.

By I.H. we have

$$ih : \underbrace{\mathsf{A}\ s\ b\ (l', r', idc', c', ids)\ r\ idcr\ p'}_{=:\ left\_b} \doteq \underbrace{\mathsf{A}\ s'_{\mathsf{n}}\ b'_{\mathsf{n}}\ l'_{\mathsf{p}}\ r\ idcr_2\ q'}_{=:\ right\_b},$$

where $idcr_2$ is obtained from $idcr$ by identity (5) and

$$\begin{aligned}s'_{\mathsf{n}} &:= \mathsf{nxtS}\ s\ l'\ r',\\ b'_{\mathsf{n}} &:= \mathsf{next_B}\ s\ l'\ r'\ idcr'\ c'\ ids\ b\ p',\\ l'_{\mathsf{p}} &:= \mathsf{pred}\ s\ l'\ r'\ idcr'\ c'\ ids.\end{aligned}$$

Let $idcr_1$, $idcr_3$ be obtained from $idcr_0$ by id_1, id_2, respectively. By UIP C we get

$$\mathsf{subst}\ ih\ idcr_1 \doteq idcr_3.$$

That means

$$(left\_b, idcr_1) \doteq (right\_b, idcr_3)$$

and by Principle 1 we get

$$\begin{aligned}\mathsf{A}\ s\ b\ (l, r, idcr, c, idsc)\ r_0\ idcr_0\ p &\doteq f\ left\_b\ idcr_1\\ &\doteq f\ right\_b\ idcr_3\\ &\doteq \mathsf{A}\ s_{\mathsf{n}}\ b_{\mathsf{n}}\ l_{\mathsf{p}}\ r_0\ idcr'_0\ q. \qquad \Box\end{aligned}$$

Corollary 33.

$$\mathsf{co\_}(\mathsf{A}\ s\ b\ (l, r, idcr, c, idsc)\ r_0\ idcr_0\ p) \doteq \mathsf{co\_}(\mathsf{A}\ s_{\mathsf{n}}\ b_{\mathsf{n}}\ l_{\mathsf{p}}\ r_0\ idcr'_0\ q).$$

Let $s : \mathsf{S}$, $l : \mathsf{CTSeq}\ s$, $r : \mathsf{R}$, $idcr : (\mathsf{co}\ r) \doteq (\mathsf{co\_}l)$, $c : \mathsf{C}$, $idsc : (\mathsf{st}\ c) \doteq (\mathsf{nxt}\ r)$. We define an inhabitant of

$$\mathsf{st\_}(l, r, idcr, c, idsc) \doteq \mathsf{st\_}(\mathsf{pred}\ s\ l\ r\ idcr\ c\ idsc) \tag{6}$$

by $\mathsf{refl}\ (\mathsf{nxt}\ r)$ according to the shape of l. Again definition by cases is necessary to define this inhabitant. The following lemma will be our main tool to prove all desired properties of T. Roughly speaking it says that we get the same trees whether we take the subtree following the tree at b along the path $(l, r, idcr, c, idsc)$ or do one step from b along this sequence (get a new b_{n}) and following the tree at b_{n} along the path obtained from $(l, r, idcr, c, idsc)$ by pred above.

Lemma 34 (*Main Lemma*). *Let $s, l, r, c, b, idcr, idsc, p, q$ as well as $s_{\mathsf{n}}, b_{\mathsf{n}}, l_{\mathsf{p}}$ be as in Proposition 32. Then*

$$\mathsf{T}'_{\mathsf{Rep}}\ s\ b\ (l, r, idcr, c, idsc)\ p \sim \mathsf{T}_{\mathsf{Rep}}\ s_{\mathsf{n}}\ b_{\mathsf{n}}\ l_{\mathsf{p}}\ q,$$

where we obtain the left term from $\mathsf{T}_{\mathsf{Rep}}\ s\ b\ (l, r, idcr, c, idsc)\ p$ by identity (6).

Proof. We have to distinguish cases $l \rightsquigarrow (c', ids)$ and $l \rightsquigarrow (l', r', idc', c', ids)$ in order to have

$$\mathsf{T}'_{\mathsf{Rep}}\ s\ b\ (l, r, idcr, c, idsc)\ p \leftrightsquigarrow \mathsf{T}_{\mathsf{Rep}}\ s\ b\ (l, r, idcr, c, idsc)\ p.$$

However, the proof proceeds in the same way in both cases:

Let $n : \mathsf{N}$. For $n \rightsquigarrow \mathsf{zero}$ is nothing to do. Let $n \rightsquigarrow \mathsf{succ}\ m$. Let $l^+ := (l, r, idcr, c, idsc)$ and

$$\begin{aligned}
c_0 &:= \mathsf{co\_}(\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l^+\ p),\\
c_1 &:= c,\\
c_2 &:= \mathsf{co\_}l_{\mathsf{p}},\\
c_3 &:= \mathsf{co\_}(\mathsf{T}_{\mathsf{Rep}}\ s_{\mathsf{n}}\ b_{\mathsf{n}}\ l_{\mathsf{p}}\ q).
\end{aligned}$$

We have

$$c_0 \doteq c_1 \doteq c_2 \doteq c_3,$$

where the first and last equations follow with identity (3) and the second with identity (5).

Now let $r_0 : \mathsf{R}$ and $idcr_0 : \mathsf{co}\, r_0 \doteq c_0$. For $i = 0, 1, 2$ we obtain elements $idcr_{i+1} : \mathsf{co}\, r_0 \doteq c_{i+1}$ from $idcr_i$ by the above identities. Further, we obtain a second element $idcr'_0 : \mathsf{co}\, r_0 \doteq c_0$ from $idcr_1$ and a second element $idcr'_3 : \mathsf{co}\, r_0 \doteq c_3$ from $idcr_0$. We have $idcr_0 \doteq idcr'_0$ and $idcr_3 \doteq idcr'_3$.[6] Let

$$\begin{aligned}
&\text{nxt\_lft} := \mathsf{next}_{\mathsf{El}}\ \_\ (\mathsf{T}_{\mathsf{Rep}}\ s\ b\ l^+\ p)\ r_0,\\
&\text{nxt\_rgt} := \mathsf{next}_{\mathsf{El}}\ \_\ (\mathsf{T}_{\mathsf{Rep}}\ s_n\ b_n\ l_p\ q)\ r_0,
\end{aligned}$$

and $ct_0 :=$ nxt_lft $idcr'_0$, $ct_1 :=$ nxt_lft $idcr_0$, $ct_2 :=$ nxt_rgt $idcr_3$, $ct_3 :=$ nxt_rgt $idcr'_3$. We have $ct_0 \doteq ct_1$ and $ct_2 \doteq ct_3$. We have to prove $ct_1 \sim_m ct_3$. Therefore it is enough to prove

$$ct_0 \sim_m ct_2.$$

Let

$$\begin{aligned}
c_{\mathsf{p}} &:= \mathsf{co\_}(\mathsf{A}\ s\ b\ l^+\ r_0\ idcr_1\ p),\\
c_{\mathsf{p}}\_id &:= \mathsf{id}^{\mathsf{S}}_{\mathsf{co\_}}(\mathsf{A}\ s\ b\ l^+\ r_0\ idcr_1\ p)\\
idc_{\mathsf{p}} &\quad \text{given by identity (5).}
\end{aligned}$$

We have

$$\begin{aligned}
&(p, \mathsf{refl}\ c_{\mathsf{p}}) : \mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s\ b\ (l^+, r_0, idcr_1, c_{\mathsf{p}}, c_{\mathsf{p}}\_id)),\\
&\quad (q, idc_{\mathsf{p}}) : \mathsf{set}(\mathsf{T}_{\mathsf{tree}}\ s_{\mathsf{n}}\ b_{\mathsf{n}}\ (\mathsf{pred}\ s\ l^+\ r_0\ idcr_1\ c_{\mathsf{p}}\ c_{\mathsf{p}}\_id))
\end{aligned}$$

[6] We do not need UIP C for this.

and

$$ct_0 \leftrightsquigarrow \mathsf{T}'_{\mathsf{Rep}}\ s\ b\ (l^+, r_0, idcr_1, c_{\mathsf{p}}, c_{\mathsf{p}}\_id)\ (p, \mathsf{refl}\, c_{\mathsf{p}}),$$

$$ct_2 \leftrightsquigarrow \mathsf{T}_{\mathsf{Rep}}\ s_{\mathsf{n}}\ b_{\mathsf{n}}\ (\mathsf{pred}\ s\ l^+\ r_0\ idcr_1\ c_{\mathsf{p}}\ c_{\mathsf{p}}\_id)\ (q, idc_{\mathsf{p}}).$$

Therefore the claim follows by I.H. applied to $s : S$, $l^+ : \mathsf{CTSeq}\ s$, $r_0 : \mathsf{R}$, $c_p : \mathsf{C}$, $b : B\ s$, $idcr_1 : \mathsf{co}\ r_0 \doteq c$, $c_p\_id : \mathsf{st}\ c_p \doteq \mathsf{nxt}\ r_0$, $(p, \mathsf{refl}\ c_{\mathsf{p}})$, (q, idc_{p}) and $m : \mathsf{N}$. □

Corollary 35. *For* $s : \mathsf{S}$, $b : B\ s$, $r : \mathsf{R}$, $idcr : (\mathsf{co}\ r) \doteq \mathsf{co}\_(\mathsf{T}\ s\ b)$, *we have*

$$(\mathsf{elim}\ s\ (\mathsf{T}\ s\ b))_{\mathsf{next_{El}}}\ r\ idcr \sim (\mathsf{Prog}\ \mathsf{T}\ s\ (g\ s\ b))_{\mathsf{next_{El}}}\ r\ idcr.$$

Proof. Apply the Main Lemma to

$$s, (c_0, ids_0), r, c_1, b, idcr, ids_1, (\mathsf{refl}\ c_0, \mathsf{refl}\ c_2), \mathsf{refl}\ c_3,$$

where

$$\begin{aligned}
c_0 &:= \mathsf{co}\_(\mathsf{T}\ s\ b),\\
ids_0 &:= \mathsf{id}^{\mathsf{S}}_{\mathsf{co}}\_(\mathsf{T}\ s\ b),\\
c_1 &:= \mathsf{co}\_(\mathsf{next_{El}}\ \_\ (\mathsf{T}\ s\ b)\ r\ idcr),\\
ids_1 &:= \mathsf{id}^{\mathsf{S}}_{\mathsf{co}}\_(\mathsf{next_{El}}\ \_\ (\mathsf{T}\ s\ b)\ r\ idcr),\\
c_2 &:= \mathsf{co}\_(\mathsf{A}\ s\ b\ (c_0, ids_0)\ r\ idcr\ (\mathsf{refl}\ c_0)),\\
c_3 &:= \mathsf{co}\_((g\ s\ b)_{\mathsf{next_{El}}}\ r\ idcr). \qquad \square
\end{aligned}$$

Note that $(\mathsf{Prog}\ \mathsf{T}\ s\ (g\ s\ b))_{\mathsf{next_{El}}}\ r\ idcr \rightsquigarrow \mathsf{T}\ (\mathsf{nxt}\ r)\ ((g\ s\ b)_{\mathsf{next_{El}}}\ r\ idcr)$.

8. Proof of the final coalgebra property

Proposition 36. *If* g *is extensional, then* T *is extensional.*

Proof. We denote the equivalence relation on B by $\equiv$ and the witness that g is extensional by ext. The proof is by natural induction. Let $s : \mathsf{S}$, $b_0, b_1 : B\ s$, $rel : b_0 \equiv b_1$, $n \rightsquigarrow \mathsf{succ}\ m : \mathsf{N}$. Let

$$\begin{aligned}
c_0 &:= \mathsf{co}\_(\mathsf{T}\ s\ b_0),\\
c_1 &:= \mathsf{co}\_(\mathsf{T}\ s\ b_1),\\
\mathrm{left\_fun} &:= (g\ s\ b_0)_{\mathsf{next_{El}}},\\
\mathrm{right\_fun} &:= (g\ s\ b_1)_{\mathsf{next_{El}}},\\
id &:= (\mathrm{ext}\ s\ b_0\ b_1\ rel)_{\mathsf{idc}} : c_0 \doteq c_1.
\end{aligned}$$

The term id gives the first component of the inhabitant we have to construct. For the second component let $r : \mathsf{R}$, $idcr : (\mathsf{co}\ r) \doteq c_0$. We have to prove

$$(\mathsf{elim}\ s\ (\mathsf{T}\ s\ b_0))_{\mathsf{next_{El}}}\ r\ idcr \sim_m (\mathsf{elim}\ s\ (\mathsf{T}\ s\ b_1))_{\mathsf{next_{El}}}\ r\ idcr',$$

where $idcr' := \mathsf{subst}\ id\ idcr$. Let $b'_0 :=$ left_fun $r\ idcr$, $b'_1 :=$ right_fun $r\ idcr'$, then $(\mathrm{ext}\ s\ b_0\ b_1\ rel)_{\mathsf{fct}}\ r\ idcr$ gives $b'_0 \equiv b'_1$ and by I.H. we have

$$\mathsf{T}\ (\mathsf{nxt}\ r)\ b'_0 \sim_m \mathsf{T}\ (\mathsf{nxt}\ r)\ b'_1.$$

The claim follows with Corollary 35. □

Lemma 37.

$$\mathsf{elim} \circ \mathsf{T} = \mathsf{Prog}\ \mathsf{T} \circ g.$$

Proof. Let $s : \mathsf{S}$, $b_0, b_1 : B\ s$, $rel : b_0 \equiv b_1$, $c_0 := ((\mathsf{elim} \circ \mathsf{T})\ s\ b_0)_{\mathsf{command}}$, $c_1 := ((\mathsf{Prog}\ \mathsf{T} \circ g)\ s\ b_1)_{\mathsf{command}}$. It is $id := ext\ s\ b_0\ b_1\ rel : c_0 \doteq c_1$. Let $n : \mathsf{N}$, $r : \mathsf{R}$ and $idcr : (\mathsf{co}\ r) \doteq c_0$. Then follows

$$((\mathsf{elim} \circ \mathsf{T})\ s\ b_0)_{\mathsf{next}_{\mathsf{El}}}\ r\ idcr \sim_n ((\mathsf{Prog}\ \mathsf{T} \circ g)\ s\ b_0)_{\mathsf{next}_{\mathsf{El}}}\ r\ idcr' \sim_n ((\mathsf{Prog}\ \mathsf{T} \circ g)\ s\ b_1)_{\mathsf{next}_{\mathsf{El}}}\ r\ idcr',$$

where $idcr' := \mathsf{subst}\ id\ idcr$ and the first relation follows by Corollary 35 and the second by the extensionality of g and $\mathsf{Prog}\ \mathsf{T}$. □

Lemma 38. *For* $\mathsf{T}' : B \to \mathsf{CT}$ *with*

$$\mathsf{elim} \circ \mathsf{T}' = \mathsf{Prog}\ \mathsf{T}' \circ g$$

we have $\mathsf{T}' = \mathsf{T}$.

Proof. Natural induction. Let $s : \mathsf{S}$, $b_0, b_1 : B\ s$, $rel : b_0 \equiv b_1$, $n \rightsquigarrow \mathsf{succ}\ m : \mathsf{N}$. Let $\mathsf{comm} : \mathsf{elim} \circ \mathsf{T}' = \mathsf{Prog}\ \mathsf{T}' \circ g$, $c_0 := ((\mathsf{elim} \circ \mathsf{T}')\ s\ b_0)_{\mathsf{command}}$, $c_1 := ((\mathsf{elim} \circ \mathsf{T})\ s\ b_1)_{\mathsf{command}}$. Then

$$id := (\mathsf{comm}\ s\ b_0\ b_1\ rel)_{\mathsf{idc}} : c_0 \doteq c_1.$$

Let $r : \mathsf{R}$ and $idcr : (\mathsf{co}\ r) \doteq c_0$, then

$$\begin{aligned}(\mathsf{T}'\ s\ b_0)_{\mathsf{next}_{\mathsf{El}}}\ r\ idcr &\sim_n \mathsf{T}'\ (\mathsf{nxt}\ r)\ ((g\ s\ b_1)_{\mathsf{next}_{\mathsf{El}}}\ r\ idcr') &(7)\\ &\sim_n \mathsf{T}\ (\mathsf{nxt}\ r)\ ((g\ s\ b_1)_{\mathsf{next}_{\mathsf{El}}}\ r\ idcr') &(8)\\ &\sim_n (\mathsf{T}\ s\ b_1)_{\mathsf{next}_{\mathsf{El}}}\ r\ idcr', &(9)\end{aligned}$$

where $idcr' := \mathsf{subst}\ id\ idcr$. Relation (7) follows by

$$(\mathsf{comm}\ s\ b_0\ b_1\ rel)_{\mathsf{fct}}\ r\ idcr,$$

relation (8) by the I.H. and the fact that $\equiv$ is reflexive and relation (9) by Corollary 35. □

Theorem 39. $\mathsf{elim} : \mathsf{CT} \to \mathsf{Prog}\quad \mathsf{CT}$ *is a final coalgebra for* Prog.

Proof. Lemmata 37 and 38. □

9. Carry over the result to the original functor of Hancock/Setzer

In this section we are going to translate the result to the original functor $\mathsf{Prog}_{\mathsf{HS}}$ of Hancock/Setzer above. We first notice that we can write an uncurried version of the functor Prog as

$$\begin{aligned}\mathsf{Prog}_{\mathsf{uc}}\ X\ s := \textstyle\sum(&p : (\mathsf{FamToPred}'\ \mathsf{st})\ s,\\ &(q : (\mathsf{FamToPred}'\ \mathsf{co})\ p_{\mathsf{fst}}) \to X\ (\mathsf{nxt}\ q_{\mathsf{fst}})).\end{aligned}$$

We can prove a final coalgebra theorem for this functor in the same way as above (this is just a rearrangement of parentheses). If $(\mathsf{S}, \mathsf{C}, \mathsf{R}, \mathsf{st}, \mathsf{co}, \mathsf{nxt})$ comes from a Hancock/Setzer interface (S, C, R, n) as described in Section 2.4, $\mathsf{Prog}_{\mathsf{uc}}\ X\ s$ rewrites is rewritten as

$$\begin{aligned}\textstyle\sum\ (&p : \textstyle\sum(sc : \sum(S, C), (\mathsf{st}\ sc) \doteq s),\\ &(q : \textstyle\sum(scr : \sum(\sum(S, C), R'), (\mathsf{co}\ scr) \doteq_{\sum(S,C)} p_{\mathsf{fst}})) \to X\ (\mathsf{nxt}\ q_{\mathsf{fst}})),\end{aligned}$$

where st and co are the first projections and R' is the uncurried version of R. We define functions

$$\mathsf{u\_hs} : \mathsf{Prog}_{\mathsf{uc}}\ X\ s \to \mathsf{Prog}_{\mathsf{HS}}\ X\ s$$

by

$$\langle\langle\langle s', c\rangle, id\rangle, f\rangle \mapsto \langle c', f'\rangle,$$

where $c' := \mathsf{subst}\ id\ c$, $f'\ r := f\ \langle\langle\langle s, c'\rangle, r\rangle, id'\rangle$ and $id' : \langle s, c'\rangle \doteq \langle s', c\rangle$ is defined by structural induction on $id : s' \doteq s$ and

$$\mathsf{hs\_u} : \mathsf{Prog}_{\mathsf{HS}}\ X\ s \to \mathsf{Prog}_{\mathsf{uc}}\ X\ s$$

by

$$\langle c, f\rangle \mapsto \langle\langle\langle s, c\rangle, \mathsf{refl}\ s\rangle, f'\rangle,$$

where $f'\ p = (\mathsf{subst}\ p_{\mathsf{snd}}\ f)\ p_{\mathsf{fst}}$.

We have $p \leftrightsquigarrow \mathsf{u\_hs}\ (\mathsf{hs\_u}\ p)$ and therefore $p \doteq \mathsf{u\_hs}\ (\mathsf{hs\_u}\ p)$. We define equivalence relations $\cong$ on $\mathsf{Prog}_{\mathsf{uc}}\ X\ s$ by

$$\langle scid_0, f_0\rangle \cong \langle scid_1, f_1\rangle :\Leftrightarrow \exists id : scid_0 \doteq scid_1.\mathsf{pointeq}\ (f_0'\ id)\ f_1,$$

where $f_0'\ id := \mathsf{subst}\ id\ f_0$ and $\mathsf{pointeq}\ (f_0'\ id)\ f_1$ expresses that $(f_0'\ id)$, f_1 are pointwise equal. By structural induction on *id* follows that we have

$$\cong\ \subset \equiv_{\mathsf{Prog}}$$

for arbitrary equivalence relations $\equiv$ on X. Further, we have:

Proposition 40.

$$p \cong \mathsf{hs\_u}\ (\mathsf{u\_hs}\ p)$$

for $p : \mathsf{Prog}_{\mathsf{uc}}\ X\ s$.

Proof. Let $p \rightsquigarrow \langle\langle\langle s', c'\rangle, ids\rangle, f\rangle$. We prove

$$\langle\langle\langle s', c'\rangle, ids\rangle, f\rangle \cong \mathsf{hs\_u}\ (\mathsf{u\_hs}\ \langle\langle\langle s', c'\rangle, ids\rangle, f\rangle)$$

by structural induction on *ids*. That means we have to prove

$$\langle\langle\langle s, c'\rangle, \mathsf{refl}\ s\rangle, f\rangle \cong \mathsf{hs\_u}\ (\mathsf{u\_hs}\ \langle\langle\langle s, c'\rangle, \mathsf{refl}\ s\rangle, f\rangle).$$

We get an inhabitant of this type by setting the first component

$$\mathsf{refl}\ \langle\langle s, c'\rangle, \mathsf{refl}\ s\rangle.$$

The second component must now have type

$$\mathsf{pointeq}\ f\ (\mathsf{hs\_u}\ \langle c, f'\rangle)_{\mathsf{snd}},$$

where $f' := \lambda r : R'\ \langle s, c'\rangle.f\ \langle\langle\langle s, c'\rangle, r\rangle, \mathsf{refl}\ \langle s, c'\rangle\rangle$. Let $sc' : \sum(S, C)$ and $\langle\langle sc, r\rangle, idsc\rangle : \sum(scr : \sum(\sum(S, C), R'), scr_{\mathsf{fst}} \doteq_{\sum(S,C)} sc')$. By structural induction on *idsc* we get

$$f\ \langle\langle sc, r\rangle, idsc\rangle \doteq (\mathsf{subst}\ idsc\ f'')\ r,$$

where $f'' := \lambda r : R'\ sc'.f\ \langle\langle sc', r\rangle, \mathsf{refl}\ sc'\rangle$. By setting $sc' = \langle s, c'\rangle$ we get

$$f\ \langle\langle sc, r\rangle, idsc\rangle \doteq (\mathsf{hs\_u}\ \langle c, f'\rangle)_{\mathsf{snd}}\ \langle\langle sc, r\rangle, idsc\rangle. \qquad \square$$

Corollary 41.

$$p \equiv_{\mathsf{Prog}} \mathsf{hs\_u}\ (\mathsf{u\_hs}\ p)$$

for $p : \mathsf{Prog}_{\mathsf{uc}}\ X\ s$.

To view $\mathsf{Prog}_{\mathsf{HS}}$ as a functor in our category we must say what $\mathsf{Prog}_{\mathsf{HS}}$ is doing on the equivalence relations $\equiv$ on X. Therefore we define

$$p \equiv_{\mathsf{Prog}_{\mathsf{HS}}} q :\Leftrightarrow (\mathsf{hs\_u}\ p) \equiv_{\mathsf{Prog}} (\mathsf{hs\_u}\ q).$$

hs_u, u_hs are extensional in respect of this relation and we have

$$\mathsf{hs\_u}(\mathsf{u\_hs}\ p) \equiv_{\mathsf{Prog}}\ p \qquad \mathsf{u\_hs}(\mathsf{hs\_u}\ q) \equiv_{\mathsf{Prog}_{\mathsf{HS}}} q,$$

i.e. $\mathsf{Prog}_{\mathsf{uc}}\ X\ s$ and $\mathsf{Prog}_{\mathsf{HS}}\ X\ s$ are isomorphic in our category. We have

$$\mathsf{UIP}\ \ \mathsf{S},\ \mathsf{UIP}\ \ \mathsf{C} \Leftrightarrow \mathsf{UIP}\ S \wedge \forall s : S.\mathsf{UIP}\ C\ s.$$

Therefore we get:

Theorem 42. *If* $\mathsf{UIP}\ S \wedge \forall s : S.\mathsf{UIP}\ C\ s$ *then* $\mathsf{u\_hs} \circ \mathsf{elim} : \mathsf{CT} \to \mathsf{Prog}_{\mathsf{HS}}\ \mathsf{CT}$ *is a final coalgebra for* $\mathsf{Prog}_{\mathsf{HS}}$.

10. Related and future work

As we have seen, working in intensional type theory becomes quite complicated. The dependency on proof objects for simple equations results in an intricate argumentation. We also needed the principle UIP for the sets S, C for our proof to go through. So, what did we gain by the result above? First of all as already mentioned the result can be seen as a justification for the rules of Hancock/Setzer if we replace definitional equality by bisimulation and have UIP for the sets S, C. We are convinced that replacing definitional identity by bisimulation is not a serious restriction as long as we are mainly interested in the behaviour of programs. Results such as those in Michelbrink/Setzer [34] that the monad rules hold should be provable with the altered rules. There is also an advantage if we want to prove facts about the behaviour of concrete interactive programs: we proved that the functor $\mathsf{Prog}_{\mathsf{HS}}$ has a final coalgebra whereas the rules of Hancock/Setzer give us a weakly final coalgebra only (uniqueness is missing). This should outweigh that concrete interactive programs are given by *extensional* functions $X \to \mathsf{Prog}\ X$ whereas in the approach of Hancock/Setzer *any* such function is sufficient. Sets S, C with UIP S, UIP C may as well be sufficient for practical work. However, from a theoretical point of view this restriction is unsatisfactory. It would be nice to improve the above result by getting rid of these conditions. However, the type theory enriched by the rules for a weakly final coalgebra as described in e.g. [34] results in far more elegant proofs. Note also that more types become definitional equal by these rules whereas two types which depend on bisimular programs do not have to be equal. Secondly a deeper analysis of the above proof and a comparison with proofs in other frameworks may shed some light on why working in intensional type theory is so hard. The same final coalgebra construction is already carried out in **ZFC** [34] and Gambino/Hyland [12] proved an initial algebra theorem in extensional type theory. The problem of representing final coalgebras in type theory was addressed by Lindström [26] for the special case of Aczel's non-well-founded sets. Lindström used an inverse-limit construction that requires extensional type theory. What can be said already is that the lack of a good concept for subsets as in set theory complicates work. Note that the subset theory discussed in Nordström et al. [35] may be of less or no help as long as we work in an intensional setting [39,38]. We think that Luo's coercive subtyping [27] may at least be a way to get crisper formulations.

There is an increasing interest in approaches to reason in dependent type theory about imperative programming, interaction, non-termination and general recursion. We would like to mention recent work of Michael Abbott, Thorsten Altenkirch, Neil Ghani and Conor McBride on containers [1–4]. The extension of a container (the result of applying the container construction functor to a container) is a special variant of our functor $\mathsf{Prog}_{\mathsf{HS}}$. More precisely a container with parameters is a state dependent interface with trivial n where the command sets do not depend on the state. The main difference to our work is that Abbott et al. work in an extensional type theory (the identity type is given by equalizers). In fact, they require their ambient category to be locally cartesian closed, with disjoint coproducts, W- and M-sets. Geuvers

[13] investigated formalizations of inductive and coinductive types in different lambda calculi, mainly extensions of the polymorphic lambda calculus. He showed that by adding a categorical notion of (primitive) recursion, recursion can be defined by corecursion and vice versa using polymorphism. Coquand proposed in [8] to add a guarded proof induction principle to type theory to reason about infinite objects. He gives a syntactical criterion to ensure that every term has a head normal form. Gimenéz [14] formalized an extension of the Calculus of Construction with inductive and coinductive types using similar ideas. In Capretta's [6] Ph.D. thesis coinductive types are added to Martin-Löf type theory with bisimulation as equality. Filliatre [11] interpreted Hoare triples for a programming language with both imperative and functional features in the Calculus of Inductive Constructions and proved a correctness result. There is ongoing work following the line initiated by Peter Hancock and Anton Setzer [15–19,23,34] on reasoning about interfaces and programs using ideas from category theory and functional programming, linear logic, game theory, refinement calculus and formal topology. Interfaces can be seen as objects in different categories and there are many interesting monads, comonads, adjoint situations and equivalences. In the author's paper [33] the notion of interfaces is generalized and simplified. With this simplified notion the relationship of interfaces to games becomes apparent. Stateless networks like the Internet are a natural application area for this simplified notion. As shown by Hancock/Hyvernat [15] interfaces (interaction structures) seen as predicate transformers give a connection to formal topology [40]. In fact every interface gives a natural example for a non-distributive topology. This gives as well a (until now rather vague) interpretation of safety and liveness properties of programs [25]. In [23,24] Hyvernat uses interfaces to give a model of linear logic.

Acknowledgements

The author wishes to thank Anton Setzer and Peter Hancock for fruitful discussions and encouragement for this work. I also thank two anonymous referees for their valuable comments and remarks. This article would be much less readable without their work. Last but not least I thank Ken Johnson for correcting my English in this article.

References

[1] M. Abbott, T. Altenkirch, N. Ghani, Categories of containers, in: A. Gordon (Ed.), Proc. of FOSSACS 2003, Lecture Notes in Computer Science, Vol. 2620, Springer, Berlin, 2003, pp. 23–38.
[2] M. Abbott, T. Altenkirch, N. Ghani, C. McBride, Derivatives of containers, in: M. Hofmann (Ed.), Typed Lambda Calculi and Applications, TLCA 2003, Lecture Notes in Computer Science, Vol. 2701, Springer, Berlin, 2003, pp. 16–30.
[3] M. Abbott, T. Altenkirch, N. Ghani, C. McBride, ∂ for data, Fund. Inform. 65 (1–2) (2005) 1–28.
[4] M.G. Abbott, Categories of containers, Ph.D. Thesis, University of Leicester, 2003.
[5] G. Barthe, V. Capretta, O. Pons, Setoids in type theory, J. Funct. Programming 13 (2) (2003) 261–293.
[6] V. Capretta, Abstraction and computation: type theory, algebraic structures, and recursive functions, Ph.D. Thesis, University of Nijmegen, 2002.
[7] C. Coquand, Agda, Internet, URL ⟨www.cs.chalmers.se/catarina/agda/⟩.
[8] T. Coquand, Infinite objects in type theory, in: H. Barendregt, T. Nipkow (Eds.), Selected Papers of the First Internat. Workshop on Types for Proofs and Programs, TYPES'93, Nijmegen, The Netherlands, 24–28 May 1993, Vol. 806, Springer, Berlin, 1994, pp. 62–78. URL ⟨citeseer.ist.psu.edu/107677.html⟩.
[9] P. Dybjer, Inductive families, Formal Aspects of Comput. 6 (1994) 440–465.
[10] P. Dybjer, A general formulation of simultaneous inductive–recursive definitions in type theory, J. Symbolic Logic 65 (2) (2000) 525–549.
[11] J.-C. Filliatre, Verifications of non-functional programs using interpretations in type theory, J. Funct. Programming 13 (4) (2003) 709–745.
[12] N. Gambino, M. Hyland, Wellfounded trees and dependent polynomial functors, in: M.C. Stefano Berardi, F. Damiani (Eds.), Types for Proofs and Programs (TYPES 2003): Third Internat. Workshop, TYPES 2003, Torino, Italy, April 30–May 4, 2003, Revised Selected Papers, Lecture Notes in Computer Science, Vol. 3085, Springer, Berlin, 2004.
[13] J. Geuvers, Inductive and coinductive types with iteration and recursion, in: B. Nordstrom, K. Petersson, G. Plotkin (Eds.), Informal Proc. 1992 Workshop on Types for Proofs and Programs, Bastad, Sweden, 1992, pp. 183–207.
[14] E. Gimenéz, Codifying guarded definitions with recursive schemes, in: Proc. 1994 Workshop on Types for Proofs and Programs, Lecture Notes in Computer Science, Vol. 996, 1994, pp. 39–59.
[15] P. Hancock, P. Hyvernat, Programming interfaces and basic topology, Annals of Pure and Applied Logic 137 (1–3) (2006) 189–239.
[16] P. Hancock, A. Setzer, The IO monad in dependent type theory, in: Electronic Proc. Workshop on Dependent Types in Programming, Göteborg, 27–28 March, 1999, URL ⟨www.md.chalmers.se/Cs/Research/Semantics/APPSEM/dtp99.html⟩.
[17] P. Hancock, A. Setzer, Interactive programs in dependent type theory, in: P. Clote, H. Schwichtenberg (Eds.), Proc. 14th Annu. Conf. of EACSL, CSL'00, Fischbau, Germany, 21–26 August 2000, Vol. 1862, Springer, Berlin, 2000, pp. 317–331, URL ⟨citeseer.ist.psu.edu/article/hancock00interactive.html⟩.
[18] P. Hancock, A. Setzer, Specifying interactions with dependent types, in: Workshop on Subtyping and Dependent Types in Programming, Portugal, 7 July 2000, 2000, electronic proceedings, URL ⟨www-sop.inria.fr/oasis/DTP00/Proceedings/proceedings.html⟩.

[19] P. Hancock, A. Setzer, Interactive programs and weakly final coalgebras in dependent type theory, in: L. Crosilla, P. Schuster (Eds.), From Sets and Types to Topology and Analysis. Towards Practicable Foundations for Constructive Mathematics, Oxford Logic Guides, Clarendon Press, 2005, URL ⟨www.cs.swan.ac.uk/∼csetzer/⟩.
[20] M. Hedberg, A coherence theorem for Martin-Löf's type theory, J. Funct. Programming 8 (4) (1998) 413–436.
[21] M. Hofmann, Extensional concepts in intensional type theory, Ph.D. Thesis, University of Edinburg, 1995.
[22] M. Hofmann, T. Streicher, A groupoid model refutes uniqueness of identity proofs, in: Proc. Ninth Symp. on Logic in Computer Science, Paris, 1994.
[23] P. Hyvernat, Predicate transformers and linear logic: yet another denotational model, in: J. Marcinkowski, J. Tarlecki (Eds.), 18th Internat. Workshop CSL 2004, Lecture Notes in Computer Science, Vol. 3210, Springer, Berlin, 2004.
[24] P. Hyvernat, Synchronous games, simulations and λ-calculus, in: D.R. Ghica, G. McCusker (Eds.), Games for Logic and Programming Languages, GaLoP (ETAPS 2005), 2005, pp. 1–15.
[25] L. Lamport, Proving the correctness of multiprocess programs, IEEE Trans. Software Eng. 3 (2) (1977) 125–143.
[26] I. Lindström, A construction of non-wellfounded sets within Martin-Löf's type theory, J. Symbolic Logic 54 (1) (1989) 57–64.
[27] Z. Luo, Coercive subtyping, J. Logic and Comput. 9 (1) (97–13) (1999) 105–130.
[28] P. Martin-Löf, Intuitionistic Type Theory, Bibliopolis, Napoli, 1984.
[29] P. Martin-Löf, Mathematics of infinity, in: P. Martin-Löf, G. Mints (Eds.), COLOG-88, Internat. Conf. on Computer Logic, Tallinn, USSR, December 1988, Proceedings, Lecture Notes in Computer Science, Vol. 417, Springer, Berlin, 1990, pp. 147–197.
[30] P. Martin-Löf, An intuitionistic theory of types, in: G. Sambin, J. Smith (Eds.), Twenty-Five Years of Constructive Type Theory, Oxford University Press, Oxford, 1998.
[31] C. McBride, Dependently typed functional programs and their proofs, Ph.D. Thesis, University of Edinburg, 1999.
[32] M. Michelbrink, Verifications of final coalgebra theorem in: Interfaces as Functors, Programs as Coalgebras—A Final Coalgebra Theorem in Intensional Type Theory, 2005, URL ⟨www.cs.swan.ac.uk/csmichel/⟩.
[33] M. Michelbrink, Interfaces as games, programs as strategies, in: J.-C. Filliatre, C. Paulin, B. Werner (Eds.), Types for Proofs and Programs (TYPES 2004), Lecture Notes in Computer Science, Vol. 3839, Springer, Berlin, 2006, pp. 215–231.
[34] M. Michelbrink, A. Setzer, State-dependent IO-monads in type theory, in: L. Birkedal (Ed.), Proc. 10th Conf. on Category Theory in Computer Science (CTCS 2004), Electronic Notes in Theoretical Computer Science, Vol. 122, Elsevier, Amsterdam, 2005, pp. 127–146.
[35] B. Nordström, K. Peterson, J.M. Smith, Programming in Martin-Löf's Type Theory: An Introduction, Clarendon Press, Oxford, 1990.
[36] E. Palmgren, On universes in type theory, in: G. Sambin, J. Smith (Eds.), Twenty-Five Years of Constructive Type Theory, Oxford University Press, Oxford, 1998.
[37] K. Peterson, A programming system for type theory, Technical Report S-412 96, Chalmers University of Technology, Göteborg, 1982.
[38] A. Salvesen, On information discharging and retrieval in Martin-Löf's type theory, Ph.D. Thesis, University of Oslo, 1989.
[39] A. Salvesen, J.M. Smith, The strength of the subset type in Martin-Löf's type theory, in: Proc. of LICS' 88, Edinburgh, 1988.
[40] G. Sambin, The Basic Picture, a structure for topology (the Basic Picture, I), 2001.
[41] A. Setzer, Proof theory of Martin-Löf type theory—an overview, Math. et Sci. Humaines 42 (165) (2004) 59–99.
[42] T. Streicher, Semantical Investigation into Intensional Type Theory, Habilitationsschrift, LMU München, 1993.