(Co)algebraic Foundations for Effectful Recursive Definitions

Sergey Goncharov, Christoph Rauch and Lutz Schröder
Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg
Email: \{Sergey.Goncharov, Christoph.Rauch, Lutz.Schroeder\}@fau.de

Abstract—A pervasive challenge in programming theory and practice are feature combinations. Here, we propose a semantic framework that combines monad-based computational effects (e.g. store, nondeterminism, random), underdefined or free operations (e.g. actions in process algebra and automata, exceptions), and recursive definitions (e.g. loops, systems of process equations). The joint treatment of these phenomena has previously led to models tending to one of two opposite extremes: extensional as, e.g., in domain theory, and intensional as in classical process algebra and more generally in universal coalgebra. Our metalanguage for effectful recursive definitions, designed in the spirit of Moggi’s computational metalanguage, flexibly combines these intensional and extensional aspects of computations in a single framework. We base our development on a notion of complete Elgot monad, whose defining feature is a parametrized uniform iteration operator satisfying natural axioms in the style of Simpson and Plotkin. We provide a mechanism of adjoining free operations to such monads by means of coref extensions, thus in particular allowing for a non-trivial semantics of non-terminating computations with free effects. Our main result states that the class of complete Elgot monads is closed under such coref extensions, which thus serve as domains for effectful recursive definitions with free operations. Elgot monads do not require the iterated computation to be guarded, and hence iteration operators are not uniquely determined by just their defining fixpoint equation. Our results however imply that they are uniquely determined as extending the given iteration in the base effect and satisfying the axioms. We discuss a number of examples formalized in our metalanguage, including (co)recursive definitions of process-algebraic operations on side-effecting processes.

I. INTRODUCTION

Following seminal work by Moggi [27], monads are widely used to represent computational effects in program semantics, and in fact in actual programming languages [40]. More recently, the computational significance of algebraic presentations of monads has been elaborated by Plotkin and Power [31], [32], [33], [34]. An instructive example is the state monad, which is generated by operations lookup and update. As a more basic example, the finite powerset monad, commonly used for modelling nondeterminism, corresponds to the theory of idempotent commutative monoids in terms of algebraic operations \(\emptyset\) (deadlock) and \(+\) (choice).

This view of effects as operations suggests a modularity principle in which one separates uninterpreted effects or free operations from their locally scoped definition in an enveloping language construct. This is, in fact, a familiar principle; e.g. the definition of an exception in some context is determined by the enveloping handling construct, and that of an action in a CCS process by the parallel processes in the context. (These examples show at the same time that uninterpreted effects are quite different from mere procedure calls.) This principle has been used in the novel programming paradigm of handling, developed by Plotkin and Pretnar [35] and used as the formal basis of the \(\text{eff}\) programming language [9]. In particular, this approach deals also with operations that fail to be algebraic in the sense identified by Plotkin and Power [31]; the most well-known example of a non-algebraic operation is, in fact, exception handling.

However, existing frameworks for effect handling, being aimed primarily at functional-imperative programming, do not provide completely satisfactory support for uninterpreted effects in reactive processes, i.e. computations combining side-effects, reactivity and recursion. To illustrate the problem, consider a side-effecting process of the form

\[
\text{while } \text{true do } \text{do } n := n + 1; \text{write } n
\]

that keeps outputting an increasing sequence of numbers using an uninterpreted effect \(\text{write}\) and, as an interpreted base effect, access to a store location \(n\). If we wish to interpret \(\text{write}\) as, say, communicating the output to some receiver process running in parallel, in a similar style as in standard process calculi, then existing semantic approaches will not help: Terms like the above will have divergence as their only behaviour, respectively will be interpreted as \(\bot\) in the domain-theoretic denotational semantics of languages with handling such as \(\text{eff}\) [9]; in other words, the denotation of non-terminating expressions will have no record of uninterpreted effects that could be handled in a compositional semantics.

It is, then, readily apparent that in a view of computations as terms constructed from (interpreted or uninterpreted) effect operators, infinite processes using uninterpreted effects correspond to infinite terms in the effect signature. Semantically, this is captured by forming, given an underlying monad \(T\) modelling the base effect and a signature of free operations, modelled as a polynomial functor \(\Delta\), the final coalgebra

\[
T_\Delta X = \nu\gamma. T(X + \Delta\gamma).
\]

Informally, computations over \(\Delta\) are defined coinductively as returning, when executed in the base monad, either a final result or a remaining computation consisting of an uninterpreted effect applied to further computations over \(\Delta\). This is a variant of the infinite resumption monad transformer \(\nu\gamma. T(- + \gamma)\), which has been used to study asynchronous
The salient feature of Elgot monads is an iteration operator

\[ f : A \rightarrow T(B + A) \quad \mapsto \quad f^\dagger : A \rightarrow TB \]

satisfying the axioms of parametrized uniform iterativity [37] (A is assumed to be finitely presentable in the definition of Elgot monads, and arbitrary in the definition of complete Elgot monads). These axioms include a fixpoint unfolding axiom, so \( f \) as above can be seen as a recursive definition of the elements of \( A \) as computations over \( B \); \( f^\dagger \) selects solutions for the corresponding recursive equations. No assumption is made regarding guardedness of \( f \), so that the relevant fixpoints need not be unique. The core mathematical result under the hood of our framework is that if \( T \) is a (strong) complete Elgot monad then so is \( T_\Delta \), and iteration extends uniquely from \( T \) to \( T_\Delta \). The proof of this fact is fairly involved and builds partly on work by Uustalu on parametrized monads [39]. This result itself appears to be of independent interest in the theory of iteration; in particular, it yields, to our best knowledge, new models of (parametrized) iterativity: starting from a monad \( T \) whose Kleisli category is suitably enriched over complete partial orders, we adjoin free operations from \( \Delta \) to obtain a monad \( T_\Delta \) that is canonically equipped with an iteration operator, even though the latter is neither a least fixed point (\( T_\Delta \) no longer carries an ordering) nor a unique fixed point (iteration in \( T_\Delta \) is unique, but only as extending iteration in \( T \)).

While the standard approach to modelling monad-style effects in denotational semantics is to consider free algebras in suitable categories of (pre)domains [31], [30], [8], [16], the use of Elgot monads detaches recursion from the base category. Instead, recursion becomes part of the notion of computation, i.e. a feature of the monad. This perspective supports a theory of side-effecting recursion that goes beyond domain-theoretic models (in which the monad typically inherits recursion from the base category) and allows working also in categories without an inherent notion of recursion, such as \( \text{Set} \) or categories of presheaves. Again, part of the motivation for this extra generality comes from the goal to model side-effecting recursion and uninterpreted effects in process algebra.

With the semantic foundations in place, we define a computational meta-language featuring iteration, uninterpreted effects, and a construct for (locally scoped recursive) effect definitions. We exploit iteration to define the semantics of effect definitions, which is essentially by iterated unfolding of definitions of uninterpreted effects, necessarily following an outermost-first strategy as computation terms may be non-wellfounded. Despite syntactic similarities, the mechanics of effect definitions is therefore quite different from what happens in effect handling. Although iteration is in general known to be weaker than recursion, a first-order recursion operator arises as a restricted case of our recursive effect definitions.

We work out a number of examples in our meta-language; in particular, we show how to emulate effect handling in the style of Plotkin and Pretnar [36] in our framework, and how recursive effect definitions can be used to program typical process algebraic constructions such as parallel composition.

II. PROGRAMING WITH DEFINABLE EFFECTS

We start off by introducing a higher-order metalanguage, where we make a clear distinction between computations and values. We thus combine the ideas of fine-grain call-by-value [24] (monad-based modeling of call-by-value) and call-by-push-value [23] (explicit coercion of computations to values and back via thunk/force).

Let us postulate a collection of basic value types \( \mathcal{V} \), intended to include data types such as integers, reals, etc.; further types \( A, B, \ldots \) are given by the grammar:

\[ A, B \ldots ::= A \in \mathcal{V} \mid 0 \mid 1 \mid A \times B \mid A + B \mid A \rightarrow B \mid [A]_\Delta \]

where \( \Delta \) ranges over effect contexts, which are comma-separated nonrepetitive lists of typed effect variables \( f : A \rightarrow B \) (corresponding to the operations of \( \text{eff} \)). The role of effect contexts is to indicate uninterpreted effects, in accordance with the call-by-push-value discipline: a computation can be suspended to a value, so that the corresponding effect information can only be stored in their type.

We fix two finite signatures \( \Sigma_v, \Sigma_e \) consisting of value operations and generic effects, respectively. In both cases, elements of the signature are typed operations of the form \( f : A \rightarrow B \). Intuitively, the operations from \( \Sigma_e \) correspond to the usual total mathematical functions over data (such as integer multiplication), while the operations from \( \Sigma_v \) act over computations (such as nondeterminism) and serve as an ultimate source of all possible sorts of effects. On top of these data we introduce a term language for values and computations, respectively formalized by judgements of the form

\[ \Gamma \vdash_v t : A \quad \Delta ; \Gamma \vdash_e p : B. \]

In both cases \( \Gamma \) is a variable context, i.e. a nonrepetitive list of pairs \( x_i : A_i \), where \( x_i \) is a variable of type \( A_i \). Computation terms additionally depend on an effect context \( \Delta \). Well-formed terms of both types are defined by the term formation rules in Fig. 1. The judgment \( \Delta ; \Gamma \vdash_e p : A \) resolves either to \( \Gamma \vdash_v p : A \) or to \( \Delta ; \Gamma \vdash_e p : A \) consistently inside the rule. The mutually inverse operations thunk and force are used for representing computations as values and vice versa. Instead of force \( p \) we often write ‘\( p \) for the sake of brevity.

We briefly comment on the standard language features. The coproduct \( 2 = 1 + 1 \) represents truth values, and its case-operator corresponds to conditional branching. The operations \( \text{ret} t \) of returning a value \( t \) and \( \text{do} x \leftarrow p ; q \) of sequencing a computation \( p \) with a computation \( q \) (with the outcome of \( p \) bound to \( x \)) stem from Moggi’s computational metalanguage [27]. The iteration construct \( (\text{iter} \ p) \) loops over \( q : A + B \) (which may depend on \( x \)), passing the return value of \( q \) to the next iteration via \( x \) if it is in \( B \), and finishing
if it is in \( A \). Using iteration, one easily encodes a more conventional-looking while operator \((\text{init } x \gets p \text{ while } b \text{ do } q)\) where \( p : A \) is an initialization command, \( b : 2 \) is the loop condition and \( q : A \) is the loop body. Note here that supporting 

unguarded iteration is what enables us to have a standard unrestricted loop construct: If we had only guarded iteration, we would need to ensure that loop bodies are guarded in a suitable sense, e.g. if the loop body uses an uninterpreted effect \( f \), it would need to start with a call to \( f \).

The crucial constructs of our language concern effect management. We emphasize the design decision of using generic effects instead of algebraic operations throughout. Recall that algebraic operations are natural transformations \( T B \rightarrow T A \) satisfying certain additional properties, and bijectively correspond to computations of type \( A \rightarrow B \), called generic effects [32]. In the above-mentioned example of idempotent commutative monoids, the algebraic operations \( \emptyset : T 0 \rightarrow T 1 \) and \( + : T 2 \rightarrow T 1 \) are equivalently represented by generic effects \( \text{void} : 1 \rightarrow 0 \) and \( \text{toss} : 1 \rightarrow 2 \). Given \( p : C \) and \( q : C \), we have

\[
\emptyset = \text{do } \text{‘void(⋆)’}; \text{ret } *, \quad p + q = \text{do } x \leftarrow \text{toss(⋆)}; \text{case } x \text{ of } \text{inl } * \mapsto p; \text{inr } * \mapsto q,
\]

where, recall, \( \text{‘f(x)’} \) means force \( f(x) \). The second equality embodies the intuition according to which nondeterministic choice can be understood as a deterministic decision based on the outcome of tossing a nondeterministic coin. Our framework therefore operates with two kinds of generic effects: the operations in \( \Sigma \) are interpreted effects, while the effect variables in \( \Delta \) remain uninterpreted until they are supplied with an implementation. To this end, there are two constructs \( \text{def} \) and \( \text{defrec} \) for effect definition and recursive effect definition, respectively. Both constructs supply an effect variable \( f : A \rightarrow B \) in \( p \) with an implementation using a defining term \( r \). This term depends on a variable \( x : A \) and on a continuation \( v : B \rightarrow [C]_{\Delta,f} \), and abstracting successively over \( x \) and over \( v \) yields the type \( (B \rightarrow [C]_{\Delta,f}) \rightarrow (A \rightarrow [C]_{\Delta,f}) \). Roughly, the latter map is used to interpret \( f \) in \( p \). On the syntactic side we use the notation \( f(x)@v = r \), called defining clause, which morally corresponds to more suggestive, but less accurate

\[
\text{do } z \leftarrow f(x); v(z) = r.
\]

Intuitively, effect definition constructs traverse the given term \( p \) from left to right (which amounts to an outermost-first strategy in terms of the algebraic term denoted by \( p \)), search for matches with the left-hand side of the above equation, and replace them by \( r \) (of course replacing \( x \) and \( v \) with the corresponding values calculated thereby).

The difference between \( \text{def} \) and \( \text{defrec} \) is that \( \text{def} \) interprets only the outermost occurrence of \( f \), while \( \text{defrec} \) iteratively unfolds the definition of \( f \) in an outermost-first strategy until \( f \) is completely eliminated; here, recursive calls of \( f \) may be triggered both by nested occurrences of \( f \) in the original term and by occurrences of \( f \) newly introduced by the defining term. The fact that in the case of \( \text{def} \) (unlike in the case of \( \text{defrec} \)) the resulting term may still contain \( f \) is reflected in the effect context of the corresponding rule. Note that effect definitions do not reach occurrences of \( f \) that are encapsulated in values by means of functional abstraction, i.e. the constructs keep \( \Delta \) intact in \( A \rightarrow [B]_{\Delta} \). As an illustration, we can actually encode iteration using recursive effect definitions. We use a formal free effect symbol \( f \) referring to the iteration term to be defined and define it recursively by calling the corresponding unfolding rule at every iteration:

\[
\text{iter } \text{inr } x \leftarrow p; q = (\text{defrec } f(z)@v = \text{case } z \text{ of } \text{inl } y \mapsto v(y); \text{inr } y \mapsto (\text{do } z \leftarrow q; 'f(z)) \quad (*)
\]

One other important example is the operation of effect substitution, which is defined by equation:

\[
\text{p}[h/f] = \text{defrec } f(x)@v = (\text{do } y \leftarrow 'h(x); v(y)) \text{ in } p.
\]

As the substitution operation \( [t/x] \) refers to the variable context \( \Gamma \), the effect substitution operation \( \{h/f\} \) refers to the effect context \( \Delta \). Note however, that the latter is self-referential in the sense that if \( f \) occurs in \( h \), then \( h \) will be substituted recursively until the fixpoint is reached.

In the above examples, the continuation is used conservatively, i.e. just executed at the end of the handling term; slightly more imaginative uses are found in Section V.

Standard properties of typed term calculi include invariance under structural transformations of variable contexts \( \Gamma \) such as weakening and permutation. This is easy to establish for the rules in Fig. I by straightforward induction over the derivations (cf. [14]). Analogous properties of effect context are less obvious, but nevertheless true:

**Proposition 1.** Let \( \Gamma, \Gamma' \) be variable contexts and let \( \Delta, \Delta' \) be effect contexts such that any symbol from \( \Gamma \) occurs in \( \Gamma' \) and analogously for \( \Delta \) and \( \Delta' \) (hence \( |\Gamma'| \geq |\Gamma| \) and \( |\Delta'| \geq |\Delta| \)). Then \( \Delta; \Gamma \vdash v;c : A \) implies \( \Delta'; \Gamma' \vdash v;c : A \).

### III. Monads for Iteration:

**CPO-enrichment and Costrategy Extensions**

We now proceed to set up the technical context for interpreting the language of simple programs. According to Moggi [27], a model for the metalanguage constructs \( \text{ret} \) and \( \text{do} \) must include a Cartesian category \( C \) and a strong monad \( T \) over \( C \). To cope with coproducts and function types we require moreover that \( C \) is bicartesian closed, i.e. Cartesian closed with finite coproducts. We fix such a \( C \) throughout, and denote the class of objects of \( C \) by \( |C| \). We denote injections into binary coproducts by \( \text{inl} : A \rightarrow A + B, \text{inr} : B \rightarrow A + B, \) and projections from binary products by \( \text{pr}_1, \text{pr}_2 \); pairing is denoted by \( \langle x, y \rangle \), and copairing of \( f : A \rightarrow C, g : B \rightarrow C \) by \( [f, g] : A + B \rightarrow C \). Moreover, the Cartesian closed structure induces standard mutually inverse transformations \( \text{curry} : ((X \times Y) \rightarrow Z) \rightarrow (X \rightarrow (Y \rightarrow Z)) \) and \( \text{uncurry} : (X \rightarrow (Y \rightarrow Z)) \rightarrow ((X \times Y) \rightarrow Z) \).

Recall that every bicartesian category is distributive [13], i.e. the canonical
map $Z \times Y + Z \times X \to Z \times (Y + X)$ is an isomorphism, with inverse

$$\text{dist} : Z \times (Y + X) \to Z \times Y + Z \times X.$$ 

Distributivity essentially allows for using context variables in case expressions, i.e. in copairing.

Recall that a monad over $C$ can be given by a Kleisli triple $(T, \eta, \_\eta)$ where $T$ is an endomap of $[C]$, the unit $\eta$ is a family of morphisms $\eta_X : X \to TX$, and Kleisli lifting $\_\eta$ maps $f : X \to TY$ to $f^* : TX \to TY$, subject to the equations

$$\eta^* = \text{id} \quad f^* \eta = f \quad (f^* g)^* = f^* g^*.$$ 

This is equivalent to the presentation in terms of an endofunctor $T$ with natural transformations unit and multiplication. A monad is strong if it is equipped with a natural transformation $\tau_{X,Y} : X \times TY \to T(X \times Y)$ called strength, subject to a number of coherence conditions (e.g. [27]). Strength enables interpreting programs over more than one variable, and allows for internalization of the Kleisli lifting, thus legitimating expressions like $\lambda x. (f(x))^* : X \to (TY \to TZ)$ for $f : X \to (Y \to TZ)$, which essentially encodes $\text{curry}(\text{uncurry}(f)^* \circ \tau)$. On Set, every monad is strong [22]. Henceforth we shall use the term ‘monad’ to mean ‘strong monad’ unless explicitly stated otherwise.

The standard intuition for a monad $T$ is to think of $TX$ as the set of terms in some algebraic theory, with variables taken from $X$. In this view, the unit converts variables into terms, and a Kleisli lifting $f^*$ applies a substitution $f : X \to TY$ to terms over $X$. In our setting, the ‘terms’ featuring here are often infinite; nevertheless, we sometimes call them algebraic terms for distinction from the terms in our metalanguage.

The Kleisli category $C_T$ of a monad $T$ has the same objects as $C$, and $C$-morphisms $X \to TY$ as morphisms $X \to Y$. The identity on $X$ in $C_T$ is $\eta_X$; and the Kleisli composite of $f : X \to TY$ and $g : Y \to TZ$ is $g^* \circ f$. A monad $T$ has rank $\kappa$ if it preserves $\kappa$-filtered colimits. On Set this condition intuitively means that $T$ is determined by its values on sets whose cardinality is smaller than $\kappa$.

In order to be able to interpret the iteration construct we involve monads providing suitable support for an iteration operator.

**Definition 2** (Complete Elgot monads). A monad $T$ over $C$ is an **complete Elgot monad** if it possesses an operator $\uparrow$, called iteration, sending any $f : X \to T(Y + X)$ to $f^\uparrow : \bar{X} \to TY$ satisfying the following conditions:

- **unfolding**: $[\eta, f] \uparrow \circ f = f^\uparrow$;
- **naturality**: $g^* \circ f^\uparrow = ([T \text{inl} \circ g, \eta \circ \text{inr}^* \circ f]^\uparrow$ for any $g : Y \to TZ$;
- **distributivity**: $([\eta \circ \text{inl}, h]^* \circ g)^\uparrow = ([\eta, ([\eta \circ \text{inl}, g^* \circ h])^\uparrow \circ g]$ for any $g : X \to T(Y + Z)$ and $h : Z \to T(Y + X)$;
- **codiagonal**: $(T[id, \text{inl}] \circ g)^\uparrow = (g^\uparrow)^\uparrow$ for any $g : X \to T((Y + X) + X)$;
- **uniformity**: $f \circ h = T(id + h) \circ g$ implies $f^\uparrow \circ h = g^\uparrow$ for any $g : Z \to T(Y + Z)$ and $h : Z \to X$.

Additionally, iteration must be compatible with strength in the following sense: for any $f : X \to T(Y + X)$, $\tau \circ (\text{id} \times f^\uparrow) = (T \text{dist} \sigma \circ (\text{id} \times f))^\uparrow$.

**Remark 3.** The above definition is inspired by the axioms of
parametrized uniform iterativity [37] and going back to Bloom and Ésik [12]. Adámek et al. [4] define Elgot monads by means of a slightly different system of axioms: the codiagonal and dinaturality axioms are replaced with the Bekič identity. Both axiomatizations are however equivalent, which is essentially a result about iteration theories [12, Section 6.8]. Moreover, the iteration operator in [4] is defined only for \( f : X \to T(Y + X) \) with finitely presentable \( X \), under the assumption that \( C \) is locally finitely presentable; hence our use of the term ‘complete Elgot monad’ instead of ‘Elgot monad’.

In the further development, examples of complete Elgot monads will arise either as so-called \( \omega \)-continuous monads (Definition 4) or as extensions thereof with free operations.

If \( T \) supports an iteration operator \( \nu \) then it is always possible to parametrize it with an additional argument to be carried over the recursion loop, i.e. we derive an operator \( \overline{\nu} \) sending \( f : Z \times X \to T(Y + X) \) to \( \overline{\nu} f : Z \times X \to TY \) by

\[
\overline{\nu} f = (T(\nu + \text{id}) \circ (\text{dist}) \circ \tau_{ZY+X} \circ \langle \nu f, 1 \rangle) \quad (**)
\]

We call the derived operator \( \overline{\nu} \) strong iteration.

As indicated above, an important class of examples of complete Elgot monads arises via a suitable order-enrichment of the Kleisli category.

**Definition 4** (\( \omega \)-continuous monad). An \( \omega \)-continuous monad consists of a monad \( T \) and an enrichment of the Kleisli category \( C_T \) of \( T \) over the category \( \omega \text{-Cpo} \) of \( \omega \)-complete partial orders with bottom and (nonstrict) continuous maps, satisfying the following conditions:

- strength is \( \omega \)-continuous: \( \tau(\text{id} \times \bigcup_i f_i) = \bigcup_i \tau(\text{id} \times f_i) \);
- coparing in \( C_T \) is \( \omega \)-continuous in both arguments:
  \( \bigcup_i f_i, \bigcup_i g_i = \bigcup_i [f_i, g_i] \);
- bottom elements are preserved by strength and by postcomposition in \( C_T \): \( \tau(\text{id} \times \bot) = \bot, f^* \circ \bot = \bot \).

**Example 5.** Many of the standard computational monads on \( \text{Set} \) are \( \omega \)-continuous, including nontermination (\( TX = X + 1 \)), nondeterminism (\( TX = \mathcal{P}(X) \)), and the nondeterministic state monad (\( TX = \mathcal{P}(X \times S) \) for a set \( S \) of states). On \( \omega \text{-Cpo} \), lifting (\( TX = X_\perp \)) and the various power domain monads are \( \omega \)-continuous.

**Remark 6.** As observed by Kock [22], monad strength is equivalent to enrichment over the base category. One consequence of this fundamental fact is that if \( C \) is enriched over the category \( \omega \text{-Cpo} \) of bottomless \( \omega \)-complete partial orders and \( \omega \)-continuous maps (i.e. \( C \) is an \( \mathcal{O} \)-category in the sense of Wand [41] and Smyth and Plotkin [38]), with the bicartesian closed structure enriched in the obvious sense, then \( C_T \) is also enriched over \( \omega \text{-Cpo} \), since \( T \) underlying a strong monad, is an \( \omega \text{-Cpo} \)-functor (aka locally continuous functor [38]). Then \( T \) is \( \omega \)-continuous in the sense of Definition 4 iff each \( \text{Hom}(X, TY) \) has a bottom element preserved by strength and postcomposition in \( C_T \). This allows for incorporating numerous domain-theoretic examples by taking \( C \) to be a suitable category of predomains, and \( T \), in the simplest case, the lifting monad \( TX = X_\perp \) (from which one builds more complex examples by the construction explored next).

The following result is unsurprising in the light of analogous facts known for so-called \( \omega \)-continuous theories [12, Theorem 8.2.15, Exercise 8.2.17].

**Theorem 7.** Any \( \omega \)-continuous monad is a complete Elgot monad.

**Remark 8.** Every complete Elgot monad \( T \) can express unproductive divergence as the generic effect

\[
\left( X \xrightarrow{\text{inr}} T(Y + X) \right) \quad \dagger.
\]

This computation never produces any effects, i.e. behaves like a deadlock. If \( T \) is \( \omega \)-continuous, then unproductive divergence coincides with the least element of \( \text{Hom}(X, TY) \), for which reason we use the same symbol \( \bot \) for the above morphism, but in general, there is no ordering in which unproductive divergence could be a least element. In terms of the metalinguage \( \bot \) is expressible as \( \text{iter inf x} \leftarrow \text{ret inf x} \).

In order to accommodate both iteration and free effects, we inductively construct core free extensions of Elgot monads, typically starting from a given \( \omega \)-continuous monad capturing the base effects. Given \( a, b \in [C] \) and a monad \( T \), let

\[
\left( -\right)_a = a \times -b, \quad T_a X = \nu g \cdot T(X + \left(\left( -\right)_a\right)_X);
\]

typically \( T_a X \) is the final coalgebra of \( T(X + \left(\left( -\right)_a\right)_X) \). Intuitively, \( T_a X \) is a type of possibly non-terminating computation trees, with each node consisting of a computation with side-effects specified by \( T \) that either returns a value in \( X \) or continues with one of \( a \)-many free operations combining \( b \)-many subsequent computations. Let \( \text{out} : T_a X \to T(X + \left(\left( -\right)_a\right)_X) \) be the final coalgebra structure, and let \( \text{coit}(g) : Y \to T_a X \) denote the final morphism induced by a coalgebra \( g : Y \to T(X + \left(\left( -\right)_a\right)_X) \).

It is easy to verify that \( \text{out} \) is natural in \( X \). By Lambek’s lemma, \( \text{out} \) is a natural isomorphism. Thus, \( T \) maps into \( T_a \) via \( \text{ext} = \text{out} \circ T \).

**Lemma 9.** Given a monad \( T \) and \( a, b \in [C] \), \( T^a_b \) is the functional part of a monad \( T^a_b \), with the monad structure characterized by the following properties.

1. The unit \( \eta : X \to T_a^b \) is the unique solution of \( \text{out} \circ \eta = \eta \circ \text{inr} \).
2. Given \( f : X \to T_a^b Y \), the Kleisli lifting \( f^\#: T_a^b X \to T_a^b Y \) is the unique solution of the equation \( \text{out} \circ f^\# = \text{out} \circ (\text{inr} \circ f) \).
3. Given \( f : X \to T_a^b Y \), let \( g = [f, \eta] : X + Y \to T_a^b Y \); then \( g^\# \) is a final morphism of coalgebras, namely \( g^\# = \text{coit}(\text{out} \circ (\text{inr} \circ g) \circ \text{inl}) \).
4. The strength \( \tau^\#: X \times T_a^b Y \to T_a^b(X \times Y) \) is the unique solution of \( \text{out} \circ \tau^\# = \text{out} \circ (\text{inr} \circ \tau) \circ (\text{id} \times \text{out}) \) where \( \delta : X \times (Y + (T_a^b Y)^b) \to (X + Y) + (X \times T_a^b Y)^b \) is the obvious distributivity transformation.

The proof of Lemma 9 is facilitated by the fact that \( T(X + (-)_a^b) \) can be shown to be a parametrized monad, which
implies that $T^b_a$ is a monad [39, Theorems 3.7 and 3.9]. What is new here is that we show that $T^b_a$ is, in fact, strong, i.e. that the strength defined in the last item satisfies the requisite laws [27].

Following Uustalu [39], we next introduce a notion of guardedness.

**Definition 10** (Guardedness). A morphism $f : X \rightarrow T^b_a(Y + Z)$ is guarded if there is $u : X \rightarrow T(Y + T^b_a(Y + Z)^b_a)$ such that $\eta \circ f = T([\text{inl} + \text{id}] + u) \circ u$.

Guardedness of $f : X \rightarrow T^b_a(Y + Z)$ intuitively means that any call to a computation of type $Z$ is new here is that we show that suitable category, i.e. $TX$, is guarded. That this definition indeed satisfies the axioms of complete Elgot monads.

The following results characterize $T^b_a$ within the (overlarge) category $\text{CElg}(C)$ of complete Elgot monads over $C$ and (strong) monad morphisms [26] preserving iteration in the evident sense.

**Theorem 13.** Suppose that $\text{CElg}(C)$ has an initial object $L$. Then

1) $L^b_a$ is the free complete Elgot monad over the signature functor $(-)^b_C : C \rightarrow C$;

2) For any complete Elgot monad $T$, $T^b_a$ is the coproduct of $T$ and $L^b_a$ in $\text{CElg}(C)$, with left injection $\text{ext} : T \rightarrow T^b_a$ (in particular, ext is a morphism in $\text{CElg}(C)$).

The crucial step in proving Theorem 13 is the following statement, which is interesting in its own right.

**Lemma 14.** Let $a, b \in |C|$ and let $T$, $S$ be two complete Elgot monads. Given a complete Elgot monad morphism $\rho : T \rightarrow S$ and a Kleisli morphism $u : a \rightarrow Sb$, the transformation $\xi : T^b_a \rightarrow S$ with $\xi$ defined componentwise as

$T^b_a X \overset{\eta \circ \text{inl}, \lambda \circ \text{inr}}{\longrightarrow} T(\eta \circ \text{inl}) \circ g$ then $\eta \circ f = T([\text{inl} + \text{id}] + u) \circ u$.

Proof sketch: Uustalu already proves that guarded morphisms $f$ have unique iterates $f^i$ [39, Theorem 3.11]. The key step is then to define $f^i$ for unrestricted $f$ in a consistent manner. For $f : X \rightarrow T^b_a(Y + X)$, let $\triangleright f : X \rightarrow T^b_a(Y + X)$ be the morphism $\triangleright f = T([\text{inl} + \text{id}] + u) \circ u$ (guarded by definition), where $w$ is the composed morphism

$X \overset{f}{\longrightarrow} T^b_a(Y + X)$

$\overset{\text{out}}{\longrightarrow} T((Y + X) + T^b_a(Y + X)^b_a)$

$\overset{\pi}{\longrightarrow} T((Y + T^b_a(Y + X)^b_a) + X)$

with $\pi = T([\text{inl} + \text{id}, \text{inl} \circ \text{inr}]$. That is, we guard $f$ by iterating out $f : X \rightarrow T((Y + X) + T^b_a(Y + X)^b_a)$ (in the complete Elgot monad $T^b_a$) over the middle summand of the result. It is easy to check that $\triangleright f = f$ when $f$ is guarded. We hence can define $f^i = (\triangleright f)^i$ (in $T^b_a$). Further (nontrivial) calculations show that this definition indeed satisfies the axioms of complete Elgot monads.

The existence and the exact shape of the initial complete Elgot monad $L$ mentioned in Theorem 13 depend on the properties of $C$. Recall that $C$ is hyper-extensive [2] if it has countable coproducts that are disjoint and universal (i.e. stable under pullbacks), and coproduct injections are, as subobjects, closed under countable disjoint unions. Examples include Set, $\omega \text{Cpo}$, complete metric spaces as well as all presheaf categories.

**Theorem 15.** If $C$ is hyperextensive, then the monad $L$ given by $LX = X + 1$ is the initial object in the category of complete Elgot monads over $C$.

IV. CATEGORICAL SEMANTICS OF THE METALANGUAGE

We next develop the semantics of effect definitions. We fix from now on a complete Elgot monad $T$ on a bicartesian closed category $C$.

**Assumption 16.** The final coalgebra $\nu \gamma, T(X + \sum_a a_i \times \gamma^{b_i})$ exists for all $a_1, b_1, \ldots, a_n, b_n, X \in |C|$.

It is relatively easy to see that Assumption 16 guarantees that starting with $S = T$ we can inductively form monads $S^b_a$.
as needed to model free generic effects. There are two broad classes of models satisfying Assumption 16:
- \( \mathbf{C} \) is a locally presentable category and \( \mathbb{T} \) is ranked;
- \( \mathbf{C} \) is \( \omega \text{-Cpo} \)-enriched and has colimits of \( \omega \)-chains, and
  \( \mathbb{T} \) is as in Remark 6.

Satisfaction of Assumption 16 in the first case follows from the fact that categories of coalgebras for accessible functors over locally presentable categories are again locally presentable, in particular complete [3, Exercise 2.j, Chapter 2]. This covers most of the interesting choices of base categories, such as \( \mathbf{Set}, \omega \text{-Cpo} \), various categories of predomains, and presheaf categories, as well as almost all computationally relevant monads [27], [33]. The fact that Assumption 16 is satisfied in the second case follows from Barr’s work on algebraically compact functors [6, Theorem 5.4], which also implies that the greatest fixed points of interest coincide with least fixed points. One example covered by the second clause but not by the first one is the continuation monad \( TX = (X \to R) \to R \) (which does not have rank) under the assumption that \( R \) possesses a least element.

We additionally introduce the following purely technical postulate.

**Assumption 17.** There exists an initial complete Elgot monad \( \mathbb{L} \) on \( \mathbf{C} \).

We have shown that this assumption is satisfied in the leading examples (Theorem 15). Its main purpose is to enable Proposition 18; we expect that it can eventually be made redundant by generalizing Theorem 13 to state that extensions \( \mathbb{T} \to \mathbb{T}' \) are stable under pushouts in the category of complete Elgot monads.

We now describe the semantics of the program constructs of Fig. 1. Most of these are interpreted in a standard way [14], [27]—we only comment on the cases specific to the present work. First of all we assign to every base value type \( V \in \mathcal{V} \) its semantics \( \mathbb{V} \in [\mathbf{C}] \). This extends straightforwardly to the remaining value and computation types except the following clause:

\[ [A]_\Delta = T_\Delta A \]

where \( T_\Delta \) is given by induction as follows: \( T_\Delta = T \) if \( \Delta \) is empty and \( T_{\Delta \vdash t : A} = (T_{\Delta})^n \) with \( a = A, b = B \). We give up the notational distinction between Kleisli lifting operators for different monads, and denote Kleisli lifting by \( \downarrow \) for every \( \mathbb{T}_\Delta \). For every variable context \( \Gamma = [x_1 : A_1, \ldots, x_n : A_n] \) let \( \Gamma = [A_1] \times \cdots \times [A_n] \). Theorem 12, \( \mathbb{T}_\Delta \) is the functional part of a complete Elgot monad \( \mathbb{T}_\Delta \). When required for clarification, we will index constituents of \( \mathbb{T}_\Delta \) by \( \Delta \) as well, writing, e.g., \( \eta^\Delta \) for the unit of \( \mathbb{T}_\Delta \). We interpret each value term in context \( \Gamma \vdash t : A \) as a morphism \( [\Gamma \vdash t : A] \in \text{Hom}_{\mathbb{C}}(\Gamma, A) \), and each computation term in context \( \Delta : \Gamma \vdash p : A \) as a morphism \( [\Delta; \Gamma \vdash p : A] \in \text{Hom}_{\mathbb{C}}(\Delta, T_{\Delta}(\Delta)) \), i.e. as a Kleisli morphism for \( \mathbb{T}_\Delta \). The type coercion constructs thunk and force are semantically interpreted as identities. Operations from \( \Sigma_v \) are interpreted as morphisms in the base category \( \mathbb{C} \), and operations from \( \Sigma_c \) as morphisms in the Kleisli category of \( \mathbb{T} \). The latter are converted into Kleisli morphisms for \( \mathbb{T}_\Delta \) via repeated use of \( \text{ext} \).

For an effect variable \( f : A \to B, [f]_{\Delta,f,\Delta'} : A \to T_{\Delta,f,\Delta';B} \) is the composite

\[ A \xrightarrow{\eta^\Delta \circ \text{inr}(\text{id}_A, \xi^\Delta, \xi^A)} T_\Delta(B + (T_{\Delta,f,B})_B^n) \]

\[ \text{out}^{-1} \xrightarrow{} T_{\Delta,f,B} \xrightarrow{\text{ext}} \cdots \xrightarrow{\text{ext}} T_{\Delta,f,\Delta'B} \]

where \( a = A, b = B \). Then for any \( \Gamma, \Delta \) and \( f : A \to B \in \Delta \) we put \( [\Gamma \vdash f : A \to [B]_\Delta] = [A]_{\Delta}[f]_{\Delta} \).

The interpretation of the iteration construct relies on the strong iteration operator of \( \mathbb{T}_\Delta \) (Equation (**) ); it is given by the rule

\[ \Delta; \Gamma \vdash e : p : B + A ] = \gamma_1 \]

\[ [\Delta ; \Gamma \vdash \text{iter } x \leftarrow p ; q : B ] = \lambda z . [\eta^\Delta, \lambda x . \gamma_2^\Delta(z, x)] \gamma_1(z) \]

Finally, we proceed with the semantics of the two constructs for effect definitions. For non-recursive definitions, we have

\[ [\Delta, f : A \to B ; \Gamma \vdash c \vdash p : C ] = \gamma_1 \]

\[ [\Delta, f : A \to B ; \Gamma \vdash x : A, v : B \to [C]_{\Delta}, f \vdash r : C ] = \gamma_2 \]

\[ [\Delta, f : A \to B ; \Gamma \vdash f(x)@v = r \in p : C ] = [\eta^\Delta, f, \text{id}^A] \circ \text{ext} \circ [\psi \circ (\text{id}, \gamma_1) \]

where \( \psi : \Delta \times T_{\Delta, f, C} \to T_{\Delta}(C + T_{\Delta, f, C}) \) is defined as follows. Given \( \gamma_1 : \Delta \to T_{\Delta, f, C} \) and \( \gamma_2 : \Delta \times (B \to T_{\Delta, f, C}) \to (\Delta \to T_{\Delta, f, C}) \), \( \psi \) is the composite

\[ \Gamma \xrightarrow{T_{\Delta, f, C} (\text{dist}_\circ \circ \text{id} \circ \text{out})} T_{\Delta}(\Gamma \times C) + \]

\[ \Delta \times A \times (B \to T_{\Delta, f, C}) \xrightarrow{T_{\Delta, f, C} (\text{id}_A + \gamma_2)} T_{\Delta}(C + T_{\Delta, f, C}). \]

The interpretation of recursive definitions is similar but uses strong iteration in \( \mathbb{T}_\Delta \); with \( \gamma_1, \gamma_2 \), \( \psi \) as above,

\[ [\Delta; \Gamma \vdash \text{defrec } f(x)@v = r \in p : C ] = [\psi^\Delta \circ (\text{id}, \gamma_1) \]

Thus, the difference between non-recursive and recursive definitions lies in the fact that the former returns a result in the right-hand summand \( T_{\mathbb{T}_\Delta, \Delta'} \), delivered by \( \psi \) as is, while the latter iterates \( \psi \) over \( T_{\mathbb{T}_\Delta, \Delta'} \) to end up with a result in \( T_{\mathbb{T}_\Delta, \Delta'} \).

The presented semantics is well-behaved w.r.t. structural transformations of contexts. This is straightforward for variable contexts (cf. [14]); for effect contexts, the statement relies on Theorem 13 (cf. Assumption 17).

**Proposition 18.** Let \( \Gamma, \Gamma' \) be variable contexts and let \( \Delta, \Delta' \) be effect contexts as in Proposition 1. Let \( \theta \) be the injection from \( \{1, \ldots, |\Gamma|\} \) to \( \{1, \ldots, |\Gamma'|\} \) sending \( i \) to the number of the \( i \)-th symbol of \( \Gamma \) in \( \Gamma' \) and let \( \sigma \) be the analogous map for \( \Delta \) and \( \Delta' \). Then

\[ [\Delta'; \Gamma' \vdash t : A] = \hat{\sigma} \circ [\Delta; \Gamma \vdash t : A] \circ \hat{\theta} \]

where \( \hat{\theta} : \Gamma \to \Gamma' \) is the obvious morphism induced by \( \theta \) and

\[ T_{\mathbb{T}_\Delta} = T + \sum_{f : A \to B \in \Delta} \beta^B_{A} T \xrightarrow{\hat{\sigma}} T + \sum_{f : A \to B \in \Delta'} \beta^B_{A} = T_{\mathbb{T}_\Delta'}. \]
is the obvious morphism of complete Elgot monads induced by rearranging summands (Theorem 13) as prescribed by σ.

Proposition 18 is crucial to make sense of expressions like (def \( f(x)@v = r \in p \)) with \( f : A \rightarrow B \) not in the rightmost position of the effect context of \( p \). Specifically, we put

\[
\delta^{-1} \circ [\text{def } f(x)@v = r'^{\prime} \in p']
\]

where \( \Delta \) is the effect context of \( p \), and \( p', r' \) are the same terms as \( p, r \) but in context \( \Delta' \), which is a permutation of \( \Delta \) under some \( \sigma \). Analogously, we extend the semantics of (defrec \( f(x)@v = r \in p \)).

The above constructions for effect definitions are in fact suitable for defining finite collections of effects simultaneously and possibly mutually recursively. This is because multiple generic effects can be grouped into a single generic effect. For example, for nondeterminism (Section II) the generic effects void : 1 \( \rightarrow \) 0 and toss : 1 \( \rightarrow \) 2 can be unified into

\[
[T\text{inl}]\text{void}, (T\text{inr})\text{toss}^* : 1 \rightarrow 1 + 2,
\]

which can be further used to define both void and toss with a single defining clause. For practical purposes, it is useful to have syntactic constructs implementing definitions with multiple defining clauses as follows:

\[
\Delta ; \Gamma \vdash c. p : C
\]

where \( \Delta \) contains all the \( f_i : A_i \rightarrow B_i \). The analogous generalization of recursive definitions is straightforward. We outline the semantics of the above, by encoding it in the basic metalanguage constructs, assuming \( n = 2 \) for the sake of brevity. W.l.o.g. \( \Delta \) contains an additional free effect \( f : A_1 + A_2 \rightarrow B_1 + B_2 \) not occurring in \( p, r, r_1, r_2 \). Let us form

\[
\Delta ; \Gamma \vdash c. q : C
\]

where \( \Delta \) is the effect context of \( p \);\( r \) of inl

\[
\delta^{-1} \circ [\text{def } f(x)@v = r \in p']
\]

for fresh \( f \) and for fresh \( f \). As announced in the introduction, we moreover have a first-order recursion operator as a restricted case of recursive effect definitions in which the continuation is used only at the end of the defining term, and hence can be omitted from the syntax:

\[
\Delta ; \Gamma \vdash f(x)@v = q \in p : C
\]

We define this construct by putting (for fresh \( f \))

\[
\text{letrec } w(x) = q \in p
\]

where we require that \( w \) occurs in \( p, q \) only in the form \( \text{inl}(x) \) and not under thunk (recall that defrec does not process generic effects encapsulated in values, which may cause type inconsistency in the above equation if this assumption is dropped). Thus defined, this first-order recursion operator satisfies the expected unfolding equation:

Proposition 19. For appropriately typed \( p \) and \( q \),

\[
\text{letrec } w(x) = q \in p
\]

Note that the definition of letrec via defrec implies that unfolding is applied outermost-first. We leave the verification of further equational properties [37] for further research. Unsurprisingly, iteration is also definable from recursion:

Proposition 20. For appropriately typed \( p \) and \( q \),

\[
\text{iter } x \leftarrow p; q
\]

The recursive construct (defrec \( f(x)@v = r \in p \)) can be recovered as a combination of recursion with non-recursive definitions.
Proposition 21. For suitably typed $p$ and $r$,
\[
\begin{align*}
&\text{(defrec } f(x)@v = r \text{ in } p) \\
&= \text{letrec } w(t) = (\text{def } f(x)@v = \text{'w(thunk}(r) \text{ in } (\text{force } t)) \\
in \text{ 'w(thunk}(p)).
\end{align*}
\]

V. EXAMPLES AND DISCUSSION
We proceed to discuss examples of (recursive) definitions in our metalanguage. Some of the examples have been studied in the context of effect handling [36], which is related to our framework of definable effects but exhibits different behaviour when analysed in detail; we discuss these differences, and demonstrate how to transfer these examples to our setting nevertheless. In fact we do this generically, i.e. show that the (often desirable) behaviour of effect handling can be emulated using definable effects. As an example illustrating the added benefits of our framework w.r.t. defining effects in non-terminating computations, which is not supported by current frameworks for effect handling, we present a definition of the parallel composition of side-effecting processes. For readability, we identify objects and types. In the discussion, recall that we use the phrase algebraic term to refer to representations of monadic computations as (typically infinite) terms, while by a term we continue to mean an expression of our metalanguage.

Choice. We begin by pointing out the crucial differences in operational behaviour between effect handling in the sense of Plotkin and Pretnar [36] and definable effects. The main point is that although one iteration step in the execution of a recursive effect definition (see Section IV) removes an occurrence of a free operation from an algebraic term, it may also introduce new occurrences, and will tackle the topmost remaining occurrence in the next iteration step. Contrastingly, an effect handler will recursively wrap itself around the continuation, i.e. essentially process occurrences of free operations bottom-up. For finite terms, the results obtained by bottom-up expansion of effect definitions are often preferable. As an example, consider a setting in which we wish to implement a monad, can be applied generally. E.g. one can, in the same manner as above, evaluate toss probabilistically, assuming that $v$ abbreviates the term $\text{ret}[n]$ returning a singleton list. Let $r = \text{do } x \leftarrow v(\text{inl }*); y \leftarrow v(\text{inr }*); \text{ret}(x + y)$. By applying Proposition 21, we obtain
\[
\begin{align*}
&\text{defrec } \text{toss}(z)@v = r \text{ in } p \\
&= \text{letrec } w(t) = (\text{def } \text{toss}(z)@v = \text{'w(thunk}(r) \text{ in } (\text{force } t)) \\
in \text{ 'w(thunk}(p)).
\end{align*}
\]

It is already apparent that unfolding the recursion loop further does not simplify the original term $p$—at every iteration the same number of free effect instances remains undefined. This is avoided in the bottom-up approach as follows. Consider the recursion
\[
\begin{align*}
&\text{letrec } w(t) = (\text{def } \text{toss}(z)@v = q \text{ in force } t) \\
in \text{ 'w(thunk}(p))
\end{align*}
\]

where $q$ is obtained from $r$ by guarding the continuation variable $v$ with the recursion variable $w$:
\[
\begin{align*}
&q = \text{do } x \leftarrow \text{'w(thunk}(v(\text{inl }*))); y \leftarrow \text{'w(thunk}(v(\text{inr }*)); \text{ret}(x + y).
\end{align*}
\]

The phenomenon illustrated above now disappears, as toss is eliminated from $v(\text{inl }*)$ before $v(\text{inr }*)$ is processed.

Effect evaluation and nondeterminism. The general notion of handling [36] allows imposing arbitrary equational theories on uninterpreted effects; e.g. one can require nondeterministic choice to be commutative and associative. The arising issue of handler correctness, i.e. of verifying that handlers actually satisfy such equations, has been eliminated in eff by restricting to handling free effects only. In the design of our metalanguage we follow the same principle; however, in contrast to eff, we not only allow programming effects, but also specifying them using the underlying monad $\mathbb{T}$. This is seen most clearly in the case of nondeterminism. In the strict sense, in particular with choice being commutative, the latter is clearly not programmable in a sequential language. However, if we assume that the underlying monad $\mathbb{T}$ supports a binary operation for nondeterminism $\oplus : \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{T}$ (associative, commutative and idempotent) we can simply use it to evaluate toss using effect substitution (Section II):
\[
p\{\lambda_x. (\text{ret inl } * \oplus \text{ret inr }*)/\text{toss}\}.
\]

The same principle, i.e. to evaluate an arbitrary free effect $f : A \rightarrow B$ to a particular effect encapsulated in the underlying monad, can be applied generally. E.g. one can, in the same manner as above, evaluate toss probabilistically, assuming that
Given a free generic effect \( E \) stood as raising exceptions of type \( A \), the \( \text{eff} \) construct

\[
\text{handle } p \text{ with } e \# \text{op} \ x \ v \mapsto h
\]

(a special case of \( \text{eff} \)'s much more general handling construct) corresponds roughly to the term

\[
\text{letrec } w(u) = (\text{def } e \# \text{op}(z)@v = h' \text{ in force}(u)) \\
\quad \text{in } w(\text{thunk}(p))
\]

where \( h' \) is obtained from \( h \) by replacing every subterm \( v(t) \) with \( 'w(\text{thunk}(v(t))) \). The result should be compared to Proposition 21, which expresses recursive definitions in a similar way using plain recursion and nonrecursive definitions.

As a slogan:

\[\text{Recursive effect definitions recursively wrap themselves around defining terms, while handlers recursively wrap themselves around continuations.}\]

The above encoding works under the assumption that the continuation variable \( v \) occurs in \( h \) only in the form \( 'v(t) \) and not under thunk, a natural restriction that is respected in the examples we are aware of [9].

A specific feature of \( \text{eff} \), not covered by the above is that besides the clauses \( e \# \text{op} \ x \ v \mapsto h \) one can use the clauses \( \text{ret } x \mapsto c \) to additionally transform terminal nodes of algebraic terms into \( c : D \) (which may depend on \( x \)). This in particular results in changing the return type of the term being handled to \( D \). One possible way to emulate this feature in our setting is to use a fresh operation \( f \) protecting its arguments from being defined by \( h' \):

\[
\text{letrec } w(u) = (\text{def } e \# \text{op}(z)@v = h' ; f(x)@v = v(x) \text{ in force}(u)) \\
\quad \text{in } w(\text{thunk}(\text{do } y \leftarrow p ; x \leftarrow f(y); c)).
\]

Note here that \( x \leftarrow f(y); c \) will wrap \( f \) around \( c \), so that \( w \) will unfold \( f \) before it proceeds with \( c \); but the definition of \( f \) has no recursive call to \( w \), so that \( c \) is left unchanged.

Exceptions. Given a free generic effect \( \text{throw} : E \rightarrow 0 \) understood as raising exceptions of type \( E \), the monad \( T_{\text{throw}} = T_{E}^E \) is the exception monad \( T(- + E) \). Benton and Kennedy [10] propose a handling construct (\( \text{try } x \leftarrow p \) in \( q \) unless \( h \)) in which the computation \( p : T(A + E) \) is executed and the resulting value of type \( A \) is passed to \( q : T(B + E) \) unless an exception is thrown, in which case \( h : E \rightarrow T(B + E) \) is called. This example can be easily covered by effect handlers. We can model it explicitly using a non-recursive effect definition (using a recursive effect definition would not be suitable, for if the defining term raises the exception, the latter will again be defined recursively, which is typically undesired) : \n
\[
\text{do } y \leftarrow (\text{def } \text{throw}(e)@v = (\text{do } z \leftarrow h(e); \text{ret } \text{inr} z))
\]

where \( \Delta, \text{throw} : E \rightarrow 0 ; \Gamma \vdash \_ h : E \rightarrow C, \Delta, \text{throw} : E \rightarrow 0 ; \Gamma \vdash c \_ p : A \text{ and } \Delta, \text{throw} : E \rightarrow 0 ; \Gamma, x : A \vdash \text{inl} q : C \) (the continuation \( v : 0 \rightarrow C \) carries no information).

Stream redirection. Given a program \( \Delta, \text{print} : S \rightarrow 1 ; \Gamma \vdash c \_ p ; C \) that uses a free generic effect \( \text{print} : S \rightarrow 1 \) to print out symbols from alphabet \( S \), one may wish to collect the printed symbols into an output stream. To that end consider the defining term

\[
\Delta, \text{print} : S \rightarrow 1 ; \Gamma, s : S, v : 1 \rightarrow S^{\leq \omega} \times X \vdash r : S^{\leq \omega} \times X
\]

(where \( S^{\leq \omega} \) denotes the set of finite or infinite words over \( S \)) given by

\[
r = \text{do } (t, x) \leftarrow v(s); \text{ret } ([s|t], x)
\]

where \( [s|t] \) is the word obtained from \( t \) by adding \( s \) to the front of \( t \) if \( t \) is finite and \( f \) otherwise. The desired effect of directing the output into the stream is then achieved by the program

\[
\text{defrec } \text{print}(s)@v = r \text{ in do } x \leftarrow p; \text{ret } (s, x)
\]

where \( s : S^{\leq \omega} \) is the initial state of the stream. Variations include printing both to a file and to the console, discarding the output (redirection to /dev/null), etc. Input redirection can be organized analogously.

Mutable store. A somewhat similar example is provided by mutable store, which is (in contrast to nondeterminism) a programmable effect, i.e. can be reduced to standard type-theoretic constructs. We consider a store object \( S \) and free effects \( \text{get} : 1 \rightarrow S \) and \( \text{put} : S \rightarrow 1 \) with the obvious expected behaviour. Given \( \Delta, \Gamma \vdash c : C \) with \( \text{get}, \text{put} \in \Delta \), we encode the usual semantics of \( \text{put} \) and \( \text{get} \) as state transformers with the following definitions:

\[
\text{letrec } w(s', t) = (\text{def } \text{put}(s')@v_1 = 'w(s'', v_1(*) \text{ in force}(t)) \\
\quad \text{get}(\_ @v_2 = 'w(s', v_2(s')))
\]

Here, we thread the initial state through the program, update it throughout the loop and use its current value to resolve the occurrences of \( \text{get} \) and \( \text{put} \). A similar trick is used by Kammar et al. [21] to define the store operations by recursion and shallow handling. The latter is essentially equivalent to our non-recursive definitions.

Concurrency. Free generic effects \( \text{in} : 1 \rightarrow A \) and \( \text{out} : A \rightarrow 1 \) for input and output over an alphabet \( A \) of message names resemble communication ports as formalized in CCS but can, in our framework, be used with generalized processes [17] having arbitrary additional effects besides nondeterminism and

\[\mathbb{T}\] supports (recursive) probabilistic computations, e.g. is the subdistribution monad [19].
input/output. Assume that the monad \( T \) supports nondeterministic choice +. As an example, we then define parallel composition, not covered in the literature on effect handling [35], [36], [21]. Informally, parallel composition is defined by system of equations

\[
p \parallel q = p \parallel q + p \parallel p + q \parallel q \parallel p,
\]
\[
p \parallel q = \text{def} \; \text{in}(x : 1) @ v = \text{do} \; y \leftarrow \text{in}(x); (v(y) \parallel q);
\]
\[
\text{out}(y : A) @ w = \text{do} \; x \leftarrow \text{out}(y); (w(x) \parallel q) \text{ in } p
\]
\[
p \mid q = \text{def} \; \text{in}(x : 1) @ v =
\]
\[
\text{out}(y : A) @ w = \text{def} \; (v(y)) \parallel w(x) \text{ in } q;
\]
\[
in \text{def} \; \text{in}(x : A) @ v = (w(x) \parallel v(y)) \text{ in } q\text{ in } p.
\]

Note that in presence of other effects, the synchronization \( p \mid q \) need not be symmetric, because the precedence in which \( p \) and \( q \) run to reach the corresponding synchronization points has an impact on the joint program state at the moment of synchronization. In the above example we tackle this by simply including both \( p \mid q \) and \( q \mid p \) in \( p \parallel q \), which is by no means an ultimate solution.

The above definitions can be made completely formal by turning them into defining clauses of a recursive definition, defining free generic effects corresponding to \( \parallel \), \( \mid \) and \( \cdot \). E.g. \( \parallel \) corresponds to a free generic effect \( \text{ln} : 1 \to 2 \) (with \( 2 = 1 + 1 \)), and its defining clause in a defrec expression would have the form \( \text{ln}(z) @ v = h \) where \( h \) is obtained from the right hand side of the above equation for \( \parallel \) by replacing \( p \) with \( v(\text{inl} \ast) \) and \( q \) with \( v(\text{inr} \ast) \).

In the previous definition, it is a salient point that the effect definition recursive wraps itself around the defining term rather than the continuation; e.g. after one unfolding of \( \parallel \) we want to continue unfolding definitions in the entire right hand side of the equation, not just in \( w(x) \) and \( v(y) \).

VI. Related Work

The above results benefit from extensive previous work on monad-based axiomatic iteration and from recent work on effect handling. In particular we draw on the concept of Elgot monad studied by Adamek et.al. [4]; the construction of the free Elgot monad over a functor [5] is related to Claim 1 in our Theorem 13 on complete Elgot monads. There is extensive literature on solutions of (co)recursive program schemes [7], [1], [25], [17], from which our present work differs primarily in that we do not restrict to guarded systems of equations, therefore not only achieving considerably higher generality but establishing a key prerequisite for the semantics of effect definitions. In particular, recent work by Pirog and Gibbons [29] is concerned with a quite similar effort to combine monads with (final) coalgebras in type theory and programming but is set in the framework of completely iterative monads, in which only guarded recursive equations have solutions, which are, then, unique.

Handlers of algebraic effects were introduced by Pretnar and Plotkin [35] and then reconsidered by Bauer and Pretnar [9] in the design of eff. As we demonstrate in Section V, our use of effect definitions is able to simulate effect handlers of eff but is based on a more flexible semantic framework: like Plotkin and Pretnar we interpret generic effects over an arbitrary base monad that serves as a parameter of the framework, and the semantics is not restricted to domain models.

VII. Conclusions and Future Work

We have developed semantic foundations for recursive effect definitions, thus simultaneously covering such basic computational phenomena as generic effects, effect handling and recursion. A salient feature of our work is the axiomatic approach, which allows applying the developed theory to a broad range of models from domain theory to concurrency. This is achieved by an integration of extensional (monadic) and intensional (coalgebraic) methods. Key ingredients of our framework are

- a cofree extension construction that adjoins free generic effects to a given computational monad using a final coalgebra;
- an iteration operator inherited upwards along extensions by free generic effects, satisfying the axioms of parametrized uniform iteration operators [37]; and
- a semantics of (recursive) effect definitions that employs iteration to deal with nested occurrences of free generic effects.

We have exemplified the use of these principles in the definition of a semantics for a simple computational metalanguage featuring handling constructs in the spirit of previous work on effect handling [9], [21]. We see these results as steps towards a logical framework for verification of programs with generic effects. The use of effect definitions leads to a modular decomposition of programs (into programs with free generic effects and effect definitions) that is orthogonal to the more standard decomposition by sequencing, potentially allowing for a compositional verification of generic modules. Future work is aimed at developing a generic Hoare calculus for programs with effect definitions, extending previous results [18]. Our construct for recursive effect definition can be seen as a new way to derive a restricted form of recursion from iteration; we will investigate how this principle relates to existing work by Filinski [15] and Kakutani [20] on deriving recursion from iteration in the presence of first-class continuations.

Finally, we are planning to explore the potential of coinduction and corecursion as principles for constructing effect definitions and reasoning with them. The further exploration of the equational properties of effect definitions may profit from implementing the theory in a semi-automatic proof assistant.

Acknowledgements The authors wish to thank Stefan Milius and Paul Blain Levy for useful discussions and anonymous referees for their useful feedback to the earlier versions of the paper.
REFERENCES

[1] Aczel, P., Adámek, J., Milius, S., Velebil, J.: Infinite trees and completely iterative theories: a coalgebraic view. Theoretical Computer Science 300(1–3), 1 – 45 (2003)

[2] Adámek, J., Borger, R., Milius, S., Velebil, J.: Iterative algebras: How iterative are they? Theory Appl. Cat. 19, 61–92 (2008)

[3] Adámek, J., Rosicky, J.: Locally Presentable and Accessible Categories, LMS Lect. Notes Ser., vol. 189. Cambridge Univ. Press (1994)

[4] Adámek, J., Milius, S., Velebil, J.: Equational properties of iterative monads. Inf. Comput. 208, 1306–1348 (2010)

[5] Adámek, J., Milius, S., Velebil, J.: Elgot theories: a new perspective of the equational properties of iteration. Math. Struct. Comput. Sci. 21, 417–480 (2011)

[6] Adámek, J., Milius, S.: Algebraically compact functors. J. Pure Appl. Alg. 82, 211–231 (1992)

[7] Bartels, F.: Generalised coinduction. Math. Struct. Comput. Sci. 13, 321–348 (2003)

[8] Battenfeld, L., Schröder, M., Simpson, A.: A convenient category of domains. In: Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin. ENTCS, vol. 172, pp. 69–99. Elsevier (2007)

[9] Bauer, A., Herold, M., Pretnar, M.: Programming with algebraic effects and handlers. J. Log. Algebr. Meth. Prod. (2014)

[10] Bergstra, J.A.: Exceptional syntax. J. Funct. Prog. 11, 395–410 (2001)

[11] Bergstra, J.A.: Handbook of Process Algebra. Elsevier Science Inc., New York, NY, USA (2001)

[12] Bloom, S., Ésik, Z.: Iteration Theories. Springer (1993)

[13] Cockett, R.: Introduction to distributive categories. Math. Struct. Comput. Sci. 3, 277–307 (1993)

[14] Crole, R.: Categories for Types. Cambridge University Press (1994)

[15] Filinski, A.: On the relations between monadic semantics. Theor. Comput. Sci. 21, 397–446 (1983)

[16] Filinski, A.: Representing monads. In: Proceedings of the Twenty-First Annual ACM Symposium on Principles of Programming Languages. pp. 446–457. ACM Press (1994)

[17] Filinski, A.: On the relations between monadic semantics. Theor. Comput. Sci. 375, 41–75 (2007)

[18] Goncharov, S., Schröder, M., Simpson, A.: A coincidentia apparatus for asynchronous side-effecting processes. Inf. Comput. 231, 204 – 232 (2013)

[19] Goncharov, S., Schröder, M., Simpson, A.: A relatively complete generic hoare logic for order-enriched effects. In: Logic in Computer Science, LICS 2013. pp. 273–282 (2013)

[20] Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace semantics via coinduction. In: Logical Methods in Comp. Sci. p. 2007 (2007)

[21] Kakutani, Y.: Duality between call-by-name recursion and call-by-value iteration. In: Computer Science Logic, CSL 2002. LNCS, vol. 2471, pp. 506–521. Springer (2002)

[22] Kammar, O., Lindley, S., Oury, N.: Handlers in action. In: Int. Conf. Functional Programming, ICFP 2013. pp. 145–158. ACM (2013)

[23] Kock, A.: Strong functors and monoidal monads. Arch. Math. 23, 113–120 (1972)

[24] Levy, P.B., Power, J., Thielecke, H.: Modelling environments in call-by-value programming languages. Inf. Comput. 185, 2003 (2002)

[25] Levy, P.B., Power, J., Thielecke, H.: A subsuming paradigm. In: Girard, J.Y. (ed.) TLCA. Lecture Notes in Computer Science, vol. 1581, pp. 228–242. Springer (1999)

[26] Levy, P.B., Power, J., Thielecke, H.: Modelling environments in call-by-value programming languages. Inf. Comput. 185, 2003 (2002)

[27] Moggi, E.: A modal approach to denotational semantics. In: Category Theory and Computer Science, CTCS 1991. LNCS, vol. 530, pp. 138–159. Springer (1991)

[28] Moggi, E.: Notions of computation and monads. Inf. Comput. 93, 55–92 (1991)

[29] Piróg, M., Gibbons, J.: Monads for behaviour. In: Mathematical Foundations of Programming Semantics, MFPS 2013. ENTCS, vol. 298, pp. 309 – 324 (2013)

[30] Plotkin, G.: Some varieties of equational logic. In: Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 150–156. Springer (2006)

[31] Plotkin, G., Power, J.: Adequacy for algebraic effects. In: Foundations of Software Science and Computation Structures, FoSSaCS 2001. LNCS, vol. 2030, pp. 1–24 (2001)

[32] Plotkin, G., Power, J.: Semantics for algebraic operations. In: Mathematical Foundations of Programming Semantics, MFPS 2001. ENTCS, vol. 45, pp. 332–345. Elsevier (2001)

[33] Plotkin, G., Power, J.: Notions of computation determine monads. In: Foundations of Software Science and Computation Structures, FoSSaCS 2002. LNCS, vol. 2303, pp. 342–356. Springer (2002)

[34] Plotkin, G., Power, J.: Algebraic operations and generic effects. Appl. Cat. Struct. 11, 69–94 (2003)

[35] Plotkin, G., Pretnar, M.: Handlers of algebraic effects. In: European Symposium on Programming, ESOP 2009. LNCS, vol. 5502, pp. 80–94. Springer (2009)

[36] Plotkin, G., Pretnar, M.: Handling algebraic effects. Log. Meth. Comput. Sci. 9 (2013)

[37] Simpson, A., Plotkin, G.: Complete axioms for categorical fixed-point operators. In: Logic in Computer Science, LICS 2000. pp. 30–41 (2000)

[38] Smyth, M., Plotkin, G.: The category-theoretic solution of recursive domain equations. In: Foundations of Computer Science, FOCS 1977. pp. 13–17. IEEE Computer Society (1977)

[39] Uustalu, T.: Generalizing substitution. ITA 37(4), 315–336 (2003)

[40] Waclaw, P.: How to declare an imperative. ACM Comput. Surv. 29, 240–263 (1997)

[41] Wand, M.: Fixed-point constructions in order-enriched categories. Theoret. Comput. Sci. 8, 13–30 (1979)
Proof of Proposition 1

The statement follows from the following generalization of it, easily shown by induction over the derivation length: If \( \Delta : \Gamma \vdash_{\psi/c} t : A \) is derivable then \( \Delta' : \Gamma' \vdash_{\psi'/c} t : A' \) is derivable for any \( A' \) obtained from \( A \) by replacing its subexpressions \( \mathcal{C}[\Xi] \) with \( \mathcal{C}[\Xi'] \) in such a way the \( \Xi' \) contains all the symbols from \( \Xi \).

Proof of Theorem 7

Since copairing and Kleisli composition are continuous, the endomap \( h \mapsto [\eta, h]^*f \) over a hom-set \( \text{Hom}_C(A, TB) \) is also continuous and hence the least fixpoint of it exists by Kleene fixpoint theorem. We take this fixpoint as the definition of \( f^\dagger \).

Let us verify the axioms of Elgot monads one by one. To that end we employ the following uniformity rule for continuous functionals [32]:

\[
UF = GU \quad U(\bot) = \bot \\
\frac{U(\mu F) = \mu G}{U(\mu F) = \mu G} \quad (1)
\]

- **Naturality.** This holds by definition.
- **Unfold.** In (1) take \( F(u) = [\eta, u]^*f \), \( G(u) = [\eta, u]^*[(T \ \text{inl})g, \eta \ \text{inr}]^*f \) and \( U(h) = g^*u \). By definition, \( U(\bot) = \bot \), \( \mu F = f^\dagger \), \( \mu G = ([\eta, u]^*[(T \ \text{inl})g, \eta \ \text{inr}])^\dagger \). Then we have

\[
U(F(u)) = g^*[\eta, u]^*f = [\eta, g^*u]^*[(T \ \text{inl})g, \eta \ \text{inr}]^*f = G(U(u)).
\]

Therefore, by (1), \( g^*f^\dagger = U(\mu F) = \mu G = ([T \ \text{inl})g, \eta \ \text{inr}]^*f \)

- **Dinaturality.** Let us denote \( s = [\eta \ \text{inl}, h]^*g \) and \( t = [\eta \ \text{inl}, g]^*h \). The identity in question is then \( s^\dagger = [\eta, t^\dagger]^*g \). Observe that

\[
[\eta, t^\dagger]^*g = [\eta, [\eta, t]^\dagger]^*[\eta \ \text{inl}, g]^*h]^*g \\
=[\eta, [\eta, t]^\dagger]^*[\eta \ \text{inl}, g]^*h]^*g \\
=[\eta, [\eta, t]^\dagger]^*[\eta \ \text{inl}, h]^*g,
\]

i.e. \( [\eta, t^\dagger]^*g \) satisfies the unfolding identity for \( s^\dagger \), therefore \( s^\dagger \subseteq [\eta, t^\dagger]^*g \). By symmetry we obtain \( t^\dagger \subseteq [\eta, s^\dagger]^*h \) and therefore

\[
[\eta, t^\dagger]^*g \subseteq [\eta, [\eta, s^\dagger]^*h]^*g \\
=[\eta, s^\dagger]^*[\eta \ \text{inl}, h]^*g \\
=s^\dagger
\]

We have thus shown the identity \( s^\dagger = [\eta, t^\dagger]^*g \) by mutual inclusion.

- **Codiagonal.** Recall that we are claiming that

\[
(T(id + \nabla)g)^\dagger = (g^\dagger)^\dagger
\]

for \( g : A \rightarrow T((B + A) + A) \). We first show that \( (g^\dagger)^\dagger \) is a fixpoint of the functional defining the left-hand side as a least fixpoint, thus proving \( \subseteq \). That is, we have to show that

\[
[\eta, (g^\dagger)^\dagger]^*T(id + \nabla)g = (g^\dagger)^\dagger.
\]

We proceed as follows:

\[
(g^\dagger)^\dagger = [\eta, (g^\dagger)^\dagger]^*g \quad \text{(unfolding)} \\
= [\eta, (g^\dagger)^\dagger]^*[\eta, g]^*g \quad \text{(unfolding)} \\
= [[\eta, (g^\dagger)^\dagger], [\eta, (g^\dagger)^\dagger]^*g]^*g \\
= [[\eta, (g^\dagger)^\dagger], (g^\dagger)^\dagger]^*g \quad \text{(unfolding)} \\
= [\eta, (g^\dagger)^\dagger]^*T(id + \nabla)g \\
= [\eta, (g^\dagger)^\dagger]^* T(id + \nabla)g.
\]

For the converse inequality, continuity allows us to use fixpoint induction. Recall that the right hand side is the least fixed point of the functional \( F : (A \rightarrow TB) \rightarrow (A \rightarrow TB) \) defined by

\[
F(f) = [\eta, f]^*g^\dagger,
\]
and hence the supremum of the chain \((F^i(\bot))_{i \in \mathbb{N}}\). We show by induction on \(i\) that all members of this chain are below \((T(\text{id} + \nabla) \circ g)^{\dagger}\), with trivial induction base. So let \(f \sqsubseteq (T(\text{id} + \nabla) \circ g)^{\dagger}\). We have to show

\[
[\eta, f]^* g^{\dagger} \sqsubseteq (T(\text{id} + \nabla) g)^{\dagger}.
\]

We establish this by a second fixpoint induction on the definition of \(g^{\dagger}\), again with trivial induction base. So assume that \([\eta, f]^* r \sqsubseteq (T(\text{id} + \nabla) g)^{\dagger}\), with \(r : A \rightarrow T(B + A)\); we have to show that

\[
[\eta, f]^*[\eta, r]^* g \sqsubseteq (T(\text{id} + \nabla) g)^{\dagger}.
\]

We calculate as follows:

\[
[\eta, f]^*[\eta, r]^* g = [\eta, f, [\eta, f]^* r]^* g
\]

\[
\sqsubseteq [\eta, f, (T(\text{id} + \nabla) g)^{\dagger}]^* g\quad \text{(inner IH)}
\]

\[
\sqsubseteq [\eta, (T(\text{id} + \nabla) g)^{\dagger}, (T(\text{id} + \nabla) g)^{\dagger}]^* g\quad \text{(outer IH)}
\]

\[
= [\eta, (T(\text{id} + \nabla) g)^{\dagger}] T(\text{id} + \nabla) g
\]

\[
= (T(\text{id} + \nabla) g)^{\dagger}\quad \text{(unfolding)}
\]

- Uniformity. Let \(f : A \rightarrow T(X + A), g : B \rightarrow T(X + B), G(u) = [\eta, u]^* g, F(u) = [\eta, u]^* f, \) and \(U(u) = u^* h. \) Then \(U(\bot) = \bot\) and

\[
UF(u) = ([\eta, u]^* f)^* h
\]

\[
= [\eta, u]^* (f^* h)
\]

\[
= [\eta, u]^*[\eta \text{ inl}, (T \text{ inr}) h]^* g
\]

\[
= [[\eta, u]^*[\eta \text{ inl}, [\eta, u]^* (T \text{ inr}) h]^* g
\]

\[
= [[\eta, u]^* \text{ inl}, u^* h] g
\]

\[
= [\eta, u^* h] g
\]

\[
= GU(u).
\]

Therefore by (1), \((f^\dagger)^* h = U \mu F = \mu G = g^{\dagger}\).

In the following, we will use the axioms of strength as in [26]:

\[
pr_2 = T pr_2 \tau \quad \text{(STR}_1\text{)}
\]

\[
(T \alpha) \tau = \tau (\text{id} \times \tau) \alpha \quad \text{(STR}_2\text{)}
\]

\[
\tau (\text{id} \times \eta) = \eta \quad \text{(STR}_3\text{)}
\]

\[
(\tau (\text{id} \times f))^{\dagger} \tau = \tau (\text{id} \times f^\dagger) \quad \text{(STR}_4\text{)}
\]

where \(\alpha : (X \times Y) \times TC \rightarrow X \times (Y \times TC)\) is the associativity isomorphism of products.

To prove compatibility of strength and iteration, we proceed by first showing

\[
((T \text{ dist}) \tau (\text{id} \times f))^{\dagger} \sqsubseteq \tau (\text{id} \times f^\dagger ).
\]

First observe that, for any \(g : A \rightarrow TB\),

\[
\begin{array}{ccc}
C \times (B + A) & \xrightarrow{\text{dist}} & C \times B + C \times A \\
\text{id} \times [\eta, g] & \downarrow \text{dist}^{-1} & [\eta, \tau (\text{id} \times g)] \\
C \times TB & \xrightarrow{\tau} & T(C \times B).
\end{array}
\]

This is easily checked componentwise starting from \(C \times B + C \times A\) and using the fact that by definition \(\text{dist}^{-1} = [\text{id} \times \text{inl}, \text{id} \times \text{inr}]\). Then we have

\[
\tau (\text{id} \times f^\dagger)
\]

\[
= \tau (\text{id} \times [\eta, f^\dagger]^* f)
\]

\[
= \tau (\text{id} \times [\eta, f^\dagger]^*) \text{id} \times f
\]

\[
= (\tau (\text{id} \times [\eta, f^\dagger])) \text{id} \times f
\]

\[
= \text{(STR}_4\text{)}
\]
Lemma 22. If \( T^b_a X \) exists for each \( X \), then \( T^b_a \) is a functor and out : \( T^b_a \rightarrow T(\text{Id} + (T^b_a)_a) \) is a natural transformation. For any functor \( G : \mathcal{B} \rightarrow \mathcal{C} \), out\_G : \( T^b_a G \rightarrow T(G + (T^b_a G)_a) \) is the final \( T(G + \text{Id}_a)_a \)-coalgebra in [\( \mathcal{B}, \mathcal{C} \)].

Proof: Functoriality follows from \( T^b_a \) carrying a monad structure, as shown in the proof of Theorem 12 independently of this lemma. Now \( T^b_a f = (\eta \circ f)^b \), so by the description of \( \eta \) we have

\[
\text{out } T^b_a f = [\eta \circ f, \eta \circ \text{inr}(T^b_a f)^b] \text{ out }
\]

i.e. out is natural.

To show finality, let \( \beta : F \rightarrow T(G + a \times F^b) \) be a natural transformation. We define \( f : F \rightarrow T^b_a G \) componentwise by the equation

\[
\text{out } f_X = T(GX + a \times f_X^b) \beta_X
\]

using finality of \( \text{out } : T^b_a G X \rightarrow T(GX + (T^b_a G X)_a) \). We have to show that \( f \) is natural (uniqueness is clear). So let \( g : X \rightarrow Y \); we have to show \( f_Y F g = T^b_a G g f_X \). Note that we have a \( T(GY + (-)^b)_a \)-coalgebra

\[
T(Gg + \text{Id}) \beta_X : FX \rightarrow T(GX + (FX)_a) \rightarrow T(GY + (FX)_a);
\]

we show that both \( f_Y F g \) and \( (T^b_a G g) f_X \) are final coalgebra morphisms into \( T^b_a G Y \) for \( T(G + \text{Id}) \beta_X \). On the one hand, we have

\[
\text{out } f_Y F g = T(GY + (f_Y)_a) \beta_Y F g
\]

(definition of \( f_Y \))

\[
= T(GY + (f_Y)_a) T(Gg + (Fg)_a) \beta_X
\]

(naturality of \( \beta \))

\[
= T(GY + (f_Y F g)_a) T(Gg + \text{Id}) \beta_X.
\]

On the other hand,

\[
\text{out } (T^b_a G g) f_X = T(Gg + (T^b_a G g)_a) \text{ out } f_X
\]

(naturality of out)

\[
= T(Gg + (T^b_a G g)_a) T(GX + (f_X)_a) \beta_X
\]

(definition of \( f_X \))

\[
= T(GY + ((T^b_a G g) f_X)_a) T(Gg + \text{Id}) \beta_X.
\]

Therefore, \( (\eta \circ f)^b \) is a fixed point of the functional defining \( ((T\text{dist}) \tau )^{\downarrow} \) as a least fixpoint and the inequality above holds. The converse inequality,

\[\tau(\id \times f^\dagger) \subseteq ((T\text{dist} \tau)(\id \times f))^\dagger,\]

is shown by fixpoint induction as above for the codiagonal. The base case is trivial with

\[\tau(\text{pr}_1, \bot \text{pr}_2) = \tau(\text{pr}_1, \bot \text{pr}_1) = \tau(\id, \bot) \text{pr}_1 = \bot \text{pr}_1 = \bot.\]

Assume now that \( \tau(\id \times g) \subseteq ((T\text{dist} \tau)(\id \times f))^\dagger \). We can then calculate

\[
\tau(\id \times [\eta, g]^* (\id \times f)) = (\tau(\id \times [\eta, g]))^* \tau(\id \times f) = ([\eta, \tau(\id \times g)]^* \tau(\id \times f)) \subseteq ([\eta, (T\text{dist} \tau)(\id \times f))^\dagger]^* \tau(\id \times f)
\]

\[\subseteq ([\eta, (T\text{dist} \tau)(\id \times f))^\dagger]^* \tau(\id \times f) = ([\eta, T\text{dist} \tau(\id \times f))^\dagger \tau(\id \times f) = (T\text{dist} \tau(\id \times f)^\dagger \]

which completes the proof.

\[\square\]
Proof of Lemma 9

$T(X + (-)_{\alpha}^{b})$ can be shown to be a parametrized monad, which implies that $T_{\alpha}^{b}$ underlies a monad $\mathbb{T}_{\alpha}^{b}$, with unit and Kleisli lifting uniquely characterized by the corresponding equations [39, Theorems 3.7 and 3.9]. What is missing is to show that $\mathbb{T}_{\alpha}^{b}$ is a strong monad, as we need here.

Let us first show the identity

$$g^{b} = \text{coiT}([T(id + (T_{\alpha}^{b} \text{ inr})_{\alpha}^{b}) \text{ out } g, \eta \text{ inr}]^{*} \text{ out}),$$

(3)

By definition, $g^{b}$ is the unique morphisms making the following diagram commute:

$$
\begin{array}{ccc}
T_{\alpha}^{b}(X + Y) & \xrightarrow{g^{b}} & T_{\alpha}^{b}Y \\
\text{out} & & \text{out} \\
T(X + Y + T_{\alpha}^{b}(X + Y)_{\alpha}^{b}) & \xrightarrow{[\text{out } g, \eta \text{ inr}(g^{b}_{\alpha})]^{*} \text{ out}} & T(Y + (T_{\alpha}^{b} Y)_{\alpha}^{b})
\end{array}
$$

We then have on the one hand,

$$\text{out } g^{b} = [\text{out } f, \eta^{\nu}], \eta \text{ inr}(g^{b}_{\alpha})^{*} \text{ out} \quad \text{(definition of } \xi)$$

$$= [\text{out } f, \text{out } \eta^{\nu}, \eta \text{ inr}(g^{b}_{\alpha})]^{*} \text{ out}$$

$$= [\text{out } f, \eta \text{ inl }, \eta \text{ inr}(g^{b}_{\alpha})]^{*} \text{ out} \quad \text{(definition of } \eta^{\nu})$$

and also on the other hand,

$$T(id + g^{b}_{\alpha}) [T(id + (T_{\alpha}^{b} \text{ inr})_{\alpha}^{b}) \text{ out } g, \eta \text{ inr}]^{*} \text{ out}$$

$$= [T(id + (g^{b} T_{\alpha}^{b} \text{ inr})_{\alpha}^{b}) \text{ out } f, \eta \text{ inl }, \eta \text{ inr}(g^{b}_{\alpha})]^{*} \text{ out}$$

$$= [\text{out } f, \eta \text{ inl }, \eta \text{ inr}(g^{b}_{\alpha})]^{*} \text{ out},$$

i.e. indeed $g^{b}$ satisfies the characteristic property of the final morphism (3).

We proceed to prove that $T_{\alpha}^{b}$ is strong. We define the strength $\tau^{\nu}$ as the unique final coalgebra morphism shown in the following diagram:

$$
\begin{array}{ccc}
X \times T_{\alpha}^{b} Y & \xrightarrow{(T_{\delta})\tau^{\nu}(\text{id } \times \text{ out})} & T(X \times Y + (X \times T_{\alpha}^{b} Y)_{\alpha}^{b}) \\
\tau^{\nu} & & \text{out} \\
T_{\alpha}^{b}(X \times Y) & \xrightarrow{\text{out}} & T(X \times Y + T_{\alpha}^{b}(X \times Y)_{\alpha}^{b})
\end{array}
$$

That is, $\tau^{\nu}$ is the unique solution of equation $\text{out } \tau^{\nu} = T(id + (\tau^{\nu})_{\alpha}^{b})(T_{\delta})\tau^{\nu}(\text{id } \times \text{ out})$. By Lemma 22, $\tau^{\nu}$ is a natural transformation. Let us check the axioms of strength from [26].

- (STR1) The identity $\text{pr}_{2} = (T_{\alpha}^{b} \text{ pr}_{2})\tau^{\nu}$ follows from $T_{\alpha}^{b}(!, \text{ id}) \text{ pr}_{2} = \tau^{\nu}$, since obviously $\text{pr}_{2} = (T_{\alpha}^{b} \text{ pr}_{2})T_{\alpha}^{b}(!, \text{ id}) \text{ pr}_{2}$. Since $\tau^{\nu}$ is uniquely defined by the corresponding characteristic identity, it suffices to show that $T_{\alpha}^{b}(!, \text{ id}) \text{ pr}_{2}$ satisfies the same identity. Indeed,

$$T(id + T_{\alpha}^{b}(!, \text{ id}) \text{ pr}_{2})_{\alpha}^{b}(T_{\delta})\tau^{\nu}(\text{id } \times \text{ out})$$

$$= T(!, \text{ id}) \text{ pr}_{2} + (T_{\alpha}^{b}(!, \text{ id}) \text{ pr}_{2})_{\alpha}^{b}(T_{\delta})\tau^{\nu}(\text{id } \times \text{ out})$$

$$= T(!, \text{ id}) + (T_{\alpha}^{b}(!, \text{ id})_{\alpha}^{b}) \text{ pr}_{2} \tau^{\nu}(\text{id } \times \text{ out})$$

$$= T(!, \text{ id}) + (T_{\alpha}^{b}(!, \text{ id})_{\alpha}^{b}) \text{ out } \text{pr}_{2}$$

$$= \text{out}(T_{\alpha}^{b}(!, \text{ id}) \text{ pr}_{2}).$$

- (STR2) In order to prove that $(T_{\alpha}^{b} \alpha)^{\nu} = \tau^{\nu}(\text{id } \times \tau^{\nu}) \alpha : (X \times Y) \times T_{\alpha}^{b} C \rightarrow T_{\alpha}^{b}((X \times Y) \times C)$ where $\alpha : (X \times Y) \times T_{\alpha}^{b} C \rightarrow X \times (Y \times T_{\alpha}^{b} C)$ is the obvious associativity isomorphism, we show that $(T_{\alpha}^{b} \alpha^{-1})^{\nu}(\text{id } \times \tau^{\nu}) \alpha$ satisfies the identity characterizing $\tau^{\nu}$, i.e.

$$\text{out } T_{\alpha}^{b} \alpha^{-1} \tau^{\nu}(\text{id } \times \tau^{\nu}) \alpha = T(id + (T_{\alpha}^{b} \alpha^{-1} \tau^{\nu}(\text{id } \times \tau^{\nu}) \alpha)_{\alpha}^{b})T_{\delta}\tau^{\nu}(\text{id } \times \text{ out}).$$

16
We calculate, transforming the left hand side,
\[
\text{out } T^b_a \alpha^{-1} \tau'' (\text{id } \times \tau'') \alpha
\]
\[
= T (\alpha^{-1} + \langle T^b_a \alpha^{-1} \rangle) \text{out } \tau'' (\text{id } \times \tau'') \alpha
\]
\[
= T (\alpha^{-1} + \langle T^b_a \alpha^{-1} \rangle) \\
T (\text{id } + \langle \tau'' \rangle) (T \delta) \tau (\text{id } \times \text{out}) (\text{id } \times \tau'') \alpha
\]
\[
= T (\alpha^{-1} + \langle T^b_a \alpha^{-1} \rangle) \\
T (\text{id } + \langle \tau'' \rangle) (T \delta) \tau (\text{id } \times \text{out})) \alpha
\]
\[
\text{(definition of } \tau''\text{)}
\]
\[
\text{(definition of } \tau''\text{)}
\]
\[
\text{(naturality of } \tau\text{)}
\]
We continue to transform the last part of the term:
\[
\tau (\text{id } \times (\text{id } \times \text{out})) \alpha
\]
\[
= \tau (\text{id } \times \text{id } \times \text{id } \times \text{out}) \alpha
\]
\[
= \tau (\text{id } \times \text{id } \times \text{id } \times \text{id } \times \text{out}) \alpha
\]
\[
\text{(naturality of } \alpha\text{)}
\]
\[
\text{(naturality of } \alpha\text{)}
\]
\[
\text{(} \tau \text{ strength)}
\]
(Contracting a product of identities into an identity in the last step). Summing up, it remains to show that
\[
T (\alpha^{-1} + \langle T^b_a \alpha^{-1} \rangle) T \delta T (\text{id } \times (\text{id } \times (\text{id } \times \text{id } \times \text{out})) \alpha)
\]
\[
= T (\text{id } + \langle T^b_a \alpha^{-1} \rangle) (\text{id } \times \text{id } \times \text{id } \times \text{out}) \alpha)
\]
\[
\text{(naturality of } \alpha\text{)}
\]
\[
\text{(naturality of } \alpha\text{)}
\]
This will follow once we show that
\[
\delta (\text{id } \times (\text{id } \times (\text{id } \times \text{id } \times \text{out})) \alpha) = (\alpha + (\text{id } \times \text{id } \times \text{id } \times \text{id } \times \text{out}) \alpha) \delta.
\]
We calculate
\[
\delta (\text{id } \times (\text{id } \times (\text{id } \times \text{id } \times \text{out})) \alpha)
\]
\[
= \delta (\text{id } \times (\text{id } \times (\text{id } \times \text{id } \times \text{out})) \alpha)
\]
\[
= (\text{id } \times (\text{id } \times (\text{id } \times \text{id } \times \text{out})) \alpha)
\]
\[
= (\alpha + (\text{id } \times \text{id } \times \text{id } \times \text{id } \times \text{out}) \alpha) \delta
\]
\[
\text{(naturality of } \delta\text{)}
\]
Here, we use the obvious identity
\[
\delta (\text{id } \times \text{id } \times \text{id } \times \text{id } \times \text{out}) \alpha = (\alpha + \alpha^b) \delta
\]
in the last step, which is easily proved by reasoning in the internal language of a bicartesian closed category.

• (\text{STR}_3) In order to obtain the identity \( \tau'' (\text{id } \times \eta'') = \eta'' \), we show that the left hand side satisfies the characteristic equation for \( \eta'' \), i.e. \( \text{out } \tau'' (\text{id } \times \eta'') = \eta \text{ inl} \). Indeed,
\[
\text{out } \tau'' (\text{id } \times \eta'') = T (\text{id } + \langle \tau'' \rangle) (T \delta) \tau (\text{id } \times \eta'') (\text{id } \times \text{inl})
\]
\[
= T (\text{id } + \langle \tau'' \rangle) (T \delta) \eta (\text{id } \times \text{inl})
\]
\[
\text{(definition of } \tau''\text{)}
\]
\[
\text{(definition of } \eta''\text{)}
\]
\[
\text{(STR}_3\text{ for } \tau\text{)}
\]
\[
\text{(naturality of } \eta\text{)}
\]
\[
\text{STR}_4\text{ Given } f : X \rightarrow T^b_a Z, \text{ we show that } (\tau'' (\text{id } \times f)) \delta \tau'' = \tau'' (\text{id } \times f \delta \tau''), \text{ which generalizes the corresponding identity in } [28] \text{ under } f = \text{id}. \text{ Let } g = [f, \eta''] \text{ and let us show first that } (\tau'' (\text{id } \times g)) \delta \tau'' = \tau'' (\text{id } \times g \delta \tau''). \text{ This implies the identity for } f \text{ as follows:}
\]
\[
(\tau'' (\text{id } \times f)) \delta \tau'' = (\tau'' (\text{id } \times g)) (\text{id } \times \text{inl}) \delta \tau''
\]
\[
= (\tau'' (\text{id } \times g)) \delta T^b_a (\text{id } \times \text{inl}) \tau''
\]
\[
= (\tau'' (\text{id } \times g)) \delta \tau'' (\text{id } \times T^b_a \text{inl})
\]
\[
\text{(naturality of } \tau''\text{)}
\]
\[
\tau''(\text{id} \times g^b) = \tau''(\text{id} \times T_a^b \text{inl}) = \tau''(\text{id} \times g^b T_a^b \text{inl}) = \tau''(\text{id} \times f^b).
\]

By Lemma 9 and by definition of \(\tau''\), both \(g^b\) and \(\tau''\) are final morphisms from suitable coalgebras. By composing the corresponding commutative squares we obtain the following diagram:

\[
\begin{array}{c}
X \times T_a^b Y \xrightarrow{\text{id} \times (T_a^b \text{out})} X \times T_a^b Y + T_a^b Z \xrightarrow{T \delta} T(X \times Z + (X \times T_a^b Y)^b) \\
\end{array}
\]

from which we conclude that
\[
\tau''(\text{id} \times g^b) = \text{coit}(T \delta)(\text{id} \times [T(\text{id} + (T_a^b \text{inr})^b) \text{out}, \eta \text{inr}]^\ast \text{out}).
\]

We will be done once we show that also \((\tau''(\text{id} \times g))^b\) is a morphism from the same coalgebra to the final one, i.e. the identity
\[
(\tau''(\text{id} \times g))^b = T(\text{id} + (\tau''(\text{id} \times g))^b)(T \delta)(\text{id} \times [T(\text{id} + (T_a^b \text{inr})^b) \text{out}, \eta \text{inr}]^\ast \text{out}). \tag{4}
\]

Let us denote \(T(\text{id} + (T_a^b \text{inr})^b) \text{out} g\) by \(h\) and show that the following diagram commutes:
\[
T(X \times (Y + (T_a^b Y)^b)) \xrightarrow{T \delta} T(\text{id} \times h, \eta \text{inr})^\ast \text{out}) \tag{5}
\]

Note that \(\delta\) can explicitly be given by expression
\[
\delta(x, c) = [\lambda y. \text{inl}(x, y), \lambda z, \text{c}. \text{inr}(z, \lambda v. \langle x, c(v) \rangle)]e.
\]

Therefore,
\[
\left [(T \delta)(\text{id} \times h, \eta \text{inr})^\ast \text{out}) \right ](T \delta)
\]
\[
= (\text{id} \times h, \eta \text{inr})^\ast \text{out})
\]
\[
= (\lambda x, e). [(T \delta)(\text{id} \times h, \eta \text{inr})^\ast \text{out})]
\]
\[
= (\lambda x, e). [\lambda y. \text{(T \delta)(\text{id} \times h, \eta \text{inr})^\ast \text{out})]
\]
\[
= (\lambda x, e). [\lambda y. \text{(T \delta)(\text{id} \times h, \eta \text{inr})^\ast \text{out})]
\]
\[
= (\lambda x, e). [\lambda y. \text{(T \delta)(\text{id} \times h, \eta \text{inr})^\ast \text{out})]
\]
\[
= (\lambda x, e). [\lambda y. \text{(T \delta)(\text{id} \times h, \eta \text{inr})^\ast \text{out})]
\]

Now, we obtain (4) as follows:
\[
T(\text{id} + ((\tau''(\text{id} \times g))^b)^{\tau''(T_a^b)}) = T(\text{id} + ((\tau''(\text{id} \times g))^b)^{T_a^b})
\]
Proof of Theorem 12

We tackle Claim 1 and first show existence, i.e. we define an iteration operator on $T_0^b$ and show that it satisfies the axioms for complete Elgot monads and is consistent with iteration in $\pi$. Along the way we will establish also Claim 2.

Our notion of guardedness coincides with that of Uustalu [39], who shows that guarded morphisms $f$ have unique iterates $f^\dagger$. For any $f : X \to T_0^b(Y + X)$ (possibly not guarded) let $\triangleright f : X \to T_0^b(Y + X)$ be the morphism $\triangleright f = \text{out}^{-1} T(\text{id} + \pi)u^\dagger$ where $u$ is the composed morphism

$$X \xrightarrow{f} T_0^b(Y + X) \xrightarrow{\text{out}} T(Y + X + T_0^b(Y + X)^b) \xrightarrow{\pi} T((Y + T_0^b(Y + X)^b) + X)$$

with $\pi = T[\text{inl}, \text{inr}, \text{inl}, \text{inr}]$. Obviously, $\triangleright f$ is guarded by definition. We now define $f^\dagger = (\triangleright f)^\dagger$. To make sure that this definition is consistent we check that $\triangleright f = f$ whenever $f$ is guarded. Suppose, $\text{out} f = T(\text{id} + \pi)u$. Then $f = \text{out}^{-1} T(\text{id} + \pi)u$ and therefore

$$\triangleright f = \text{out}^{-1} T(\text{id} + \pi)(\text{out} f)^\dagger$$

$$= \text{out}^{-1} T(\text{id} + \pi)(\text{out} \text{out}^{-1} T(\text{id} + \pi)u)^\dagger$$

$$= \text{out}^{-1} T(\text{id} + \pi)(\pi T(\text{id} + \pi)u)^\dagger$$

$$= \text{out}^{-1} T(\text{id} + \pi)(T\text{inl} u)^\dagger$$

$$= \text{out}^{-1} T(\text{id} + \pi)u$$

$$= f.$$
We are left to check the axioms of Elgot monads (Definition 2).

- **Unfolding.** For any \( f : X \to T^b_a(Y + X) \) we have

\[
\begin{align*}
f^\dagger &= [\eta^\nu, f^\dagger]_{a} \triangleright f \\
&= [\eta^\nu, f^\dagger]_{a} \opi^{-1} T(\inl + \id)(\pi \out f)^\dagger \tag{definition of \( \triangleright \)} \\
&= \opi^{-1} [\eta_{\inl \out f^\dagger}, \eta \inr [\eta^\nu, f^\dagger]_{a}^b]* \tag{definition of \( \opi \)} \\
&= \opi^{-1} [\eta_{\inl \out f^\dagger}, \eta \inr [\eta^\nu, f^\dagger]_{a}^b]* (\pi \out f)^\dagger \\
&= \opi^{-1} T(\id + [\eta^\nu, f^\dagger]_{a}^b)(\pi \out f)^\dagger
\end{align*}
\]

and thus we obtain the following intermediate equation:

\[
\out f^\dagger = T(\id + [\eta^\nu, f^\dagger]_{a}^b)(\pi \out f)^\dagger.
\] (6)

Now, continuing the above calculation we obtain

\[
\begin{align*}
f^\dagger &= \opi^{-1} T(\id + [\eta^\nu, f^\dagger]_{a}^b)(\pi \out f)^\dagger \\
&= \opi^{-1} T(\id + [\eta^\nu, f^\dagger]_{a}^b)[\eta, (\pi \out f)^\dagger]\pi \out f \tag{unfolding} \\
&= \opi^{-1} [\eta(\id + [\eta^\nu, f^\dagger]_{a}^b)] \pi \out f \tag{unfolding} \\
&= \opi^{-1} [\eta(\id + [\eta^\nu, f^\dagger]_{a}^b)] \pi \out f \tag{naturality of \( \eta \)} \\
&= \opi^{-1} [\eta(\id + [\eta^\nu, f^\dagger]_{a}^b)] \pi \out f \tag{definition of \( \eta^\nu \)} \\
&= [\eta^\nu, f^\dagger]^b \opi^{-1} \out f \tag{naturality of \( \opi \)} \\
&= [\eta^\nu, f^\dagger].
\end{align*}
\]

- **Naturality.** Assume that \( h : X \to T^b_a(Y + X) \) is guarded and show that so is \([(T^b_a \inl)g, \eta \inr]^b h\) for any \( g : Y \to Z \). Let \( u \) be such that \( \out h = T(\inl + \id)u \) and let \( w = [(T^b_a \inl)g, \eta \inr] \) Then

\[
\begin{align*}
\out [(T^b_a \inl)g, \eta \inr]^b h &= [\out w, \eta \inr(w^b)]^* \out h \\
&= [\out w, \eta \inr(w^b)]^* T(\inl + \id)u \\
&= [\out w, \eta \inr(w^b)]^* u \\
&= [\out(T^b_a \inl)g, \eta \inr(w^b)]^* u \\
&= T(\inl + T^b_a \inl^b) \out g, \eta \inr(w^b)]^* u \\
&= T(\inl + \id)(T(\id + T^b_a \inl^b) \out g, \eta \inr(w^b)]^* u.
\end{align*}
\]

Now, since \( t = [(T^b_a \inl)g, \eta \inr]^b \triangleright f \) is guarded, it is the unique fixpoint of the map

\[
t \mapsto [\eta^\nu, t]^b [(T^b_a \inl)g, \eta \inr]^b \triangleright f.
\]

However, on the other hand,

\[
[\eta^\nu, g^f]^b [(T^b_a \inl)g, \eta \inr]^b \triangleright f \\
= [g, g^f]^b \triangleright f \\
= [g, g^f(\triangleright f)]^b \triangleright f \\
= g^f[\eta^\nu, (\triangleright f)]^b \triangleright f \\
= g^f f^\dagger
\]

and therefore \( t = g^f f^\dagger \).

- **Dinaturality.** Given \( g : X \to T^b_a(Y + Z) \) and \( h : Z \to T^b_a(Y + X) \), let \( s = [\eta^\nu, h^b] g : X \to T^b_a(Y + X) \), \( t = [\eta^\nu, g^b] h : Z \to T^b_a(Y + Z) \), \( w = [\eta^\nu, t^b] g : X \to T^a_a Y \). The idea is to show the identity

\[
[\eta^\nu, w]^b \triangleright s = [\eta^\nu, t^b]^b [\eta^\nu, \inl, \triangleright t]^b g
\] (7)
from which we will be able to obtain that
\[
w = [\eta^v, t]^\sharp g
= [\eta^v, [\eta^v, t]^\sharp] \triangleright t]^\sharp g
= [[\eta^v, t]^\sharp][\eta^v \text{ inl,} \triangleright t]^\sharp g
= [\eta^v, w]^\sharp \triangleright s,
\]
(i.e. that \(w\) satisfies the recursive equation uniquely identifying \(s\)) and hence \(w = s\). Let
\[
p = T((\text{id} + [\eta^v \text{ inl,} h]^b_{\alpha}) + \text{id}) : T(Y + T^h_{\alpha}(Y + Z) + Z) \to T(Y + T^h_{\alpha}(Y + X) + Z),
\]
\[
q = T((\text{id} + [\eta^v \text{ inl,} g]^b_{\alpha}) + \text{id}) : T(Y + T^h_{\alpha}(Y + X) + X) \to T(Y + T^h_{\alpha}(Y + Z) + X)
\]
and observe that
\[
(\pi \text{ out } s)^\dagger = (\pi [\text{out } \eta^v \text{ inl,} h], \eta \text{ inr } [\eta^v \text{ inl,} h]^b_{\alpha})^* \times (\pi \text{ out } g)^\dagger
= ([\eta \text{ inl } \text{id} + [\eta^v \text{ inl,} h]^b_{\alpha}], (\pi \text{ out } h))^* (\pi \text{ out } g))^\dagger
= ([\eta \text{ inl,} (\pi \text{ out } h)]^* p(\pi \text{ out } g))^\dagger.
\]  
An analogous calculation applies to \((\pi \text{ out } t)^\dagger\) and hence we obtain
\[
(\pi \text{ out } s)^\dagger = ([\eta \text{ inl,} (\pi \text{ out } h)]^* p(\pi \text{ out } g))^\dagger,
\]
\[
(\pi \text{ out } t)^\dagger = ([\eta \text{ inl,} (\pi \text{ out } g)]^* q(\pi \text{ out } h))^\dagger.
\]
Now, let us calculate the right hand side of (7):
\[
\text{out}[\eta^v, w]^\dagger \triangleright s
= \text{out}[\eta^v, w]^\dagger \text{ out}_s
= \text{out}[\eta^v, w]^\dagger \text{ out}_s
= T(\text{inl} + [\eta^v, w]^b_{\alpha})(\pi \text{ out } s)^\dagger
= T(\text{inl} + [\eta^v, w]^b_{\alpha})((\eta \text{ inl,} (\pi \text{ out } h)]^* p(\pi \text{ out } g))^\dagger
= T(\text{inl} + [\eta^v, w]^b_{\alpha})([\eta \text{ inl,} \pi(\pi \text{ out } g)]^* (\pi \text{ out } h))^\dagger)^* p(\pi \text{ out } g)
= T(\text{inl} + [\eta^v, w]^b_{\alpha})([\eta \text{ inl,} \pi(\pi \text{ out } g)]^* (\pi \text{ out } h))^\dagger)^* (\pi \text{ out } g).
\]
Now left us calculate the right hand side of (7):
\[
\text{out}[\eta^v, t]^\dagger \text{ out}[\eta^v, t]^\dagger
= [\text{out}[\eta^v, t]^\dagger, \eta \text{ inr } [\eta^v, t]^b_{\alpha}]^* \text{ out}[\eta^v, t]^\dagger
= [\text{out}[\eta^v, t]^\dagger, \eta \text{ inr } [\eta^v, t]^b_{\alpha}]^* \text{ out}[\eta^v, t]^\dagger
= [\eta \text{ inl,} \pi(\pi \text{ out } g)]^* (\pi \text{ out } h)^\dagger)^* \text{ out}[\eta^v, t]^\dagger
= \eta \text{ inl,} [\eta^v, t]^b_{\alpha}, \text{ out}[\eta^v, t]^\dagger, \eta \text{ inr } [\eta^v, t]^b_{\alpha}]^* \text{ out}[\eta^v, t]^\dagger
= [\eta \text{ inl,} \pi(\pi \text{ out } g)]^* (\pi \text{ out } h)^\dagger)^* (\pi \text{ out } g).
\]
We have thus reduced (7) to
\[
T(\text{inl} + [\eta^v, w]^b_{\alpha})([\eta \text{ inl,} \pi(\pi \text{ out } g)]^* (\pi \text{ out } h))^\dagger = [\text{out}[\eta^v, t]^\dagger, \eta \text{ inr } [\eta^v, t]^b_{\alpha}]^* \text{ out}_t.
\]  
Then, on the one hand
\[
T(\text{inl} + [\eta^v, w]^b_{\alpha})((\eta \text{ inl,} \pi(\pi \text{ out } g)]^* (\pi \text{ out } h))^\dagger
= T(\text{inl} + [\eta^v, w]^b_{\alpha} + \text{id})\eta \text{ inl,} \pi(\pi \text{ out } g)]^* (\pi \text{ out } h))^\dagger
= ([\eta \text{ inl,} \pi(\pi \text{ out } g)]^* (\pi \text{ out } h))^\dagger
\]

21
The proof is thus completed.

* Codiagonal. Let \( g : X \to T_a^b(Y + X + X) \). We shall show below that

\[
\triangleright (T_a^b(id + \nabla) \triangleright g) = \triangleright (T_a^b(id + \nabla)g).
\]

Since \( T_a^b(id + \nabla)g \) is the unique fixpoint of the map

\[
\gamma \mapsto [\eta', \gamma]^b \triangleright T_a^b(id + \nabla)g
\]

we will be done once we show that \( (g^\dagger)^\dagger \) is also a fixpoint of the same map, i.e.

\[
(g^\dagger)^\dagger = [\eta', (g^\dagger)^\dagger]^b \triangleright (T_a^b(id + \nabla)g).
\]

We denote by \( \xi : Y + X + X \to Y + X + X \) the morphism swapping two last components of the coproduct. We consider the following three cases.

1) \( T_a^b(id + \nabla)g \) is guarded. Then we obtain (11) directly as follows

\[
(g^\dagger)^\dagger = [\eta', (g^\dagger)^\dagger]^b[g'^\dagger]^a \quad \text{(unfolding)}
\]

\[
= [\eta', (g^\dagger)^\dagger]^b(\nabla')^a \quad \text{(unfolding)}
\]

\[
= [\eta', (g^\dagger)^\dagger]^b(\nabla')^a \quad \text{(unfolding)}
\]

\[
= [\eta', (g^\dagger)^\dagger]^bT_a^b(id + \nabla)g
\]

\[
= [\eta', (g^\dagger)^\dagger]^b \triangleright (T_a^b(id + \nabla)g).
\]

2) \( (T_a^b\xi)g \) is guarded. E.g. let \( (T_a^b\xi)g = \text{out}^1T(id + \nabla)u \). Then \( T_a^b(id + \nabla) \triangleright g \) is also guarded, which is certified by the following calculation:

\[
T_a^b(id + \nabla) \triangleright g
\]

\[
= T_a^b(id + \nabla) \triangleright ((T_a^b\xi) \text{out}^1T(id + \nabla)u)
\]

\[
= T_a^b(id + \nabla) \text{out}^1T(id + \nabla)(\text{out}^1T(id + \nabla)u)^\dagger
\]

\[
= T_a^b(id + \nabla) \text{out}^1T(id + \nabla)(\text{out}^1T(id + \nabla)u)^\dagger
\]

\[
= T_a^b(id + \nabla) \text{out}^1T(id + \nabla)(\text{out}^1T(id + \nabla)u)^\dagger
\]

\[
= T_a^b(id + \nabla) \text{out}^1T(id + \nabla)(\text{out}^1T(id + \nabla)u)^\dagger
\]

The proof of (11) now runs as follows:

\[
(g^\dagger)^\dagger = (((\triangleright g)^\dagger)^\dagger)^\dagger
\]

(definition of \( \dagger \))
\[= [\eta'', ((\triangleright g)^\dagger)^\dagger] \triangleright (T_a^b (id + \nabla) \triangleright g) \quad \text{(Clause 1)}
\]
\[= [\eta'', (g^\dagger)^\dagger] \triangleright (T_a^b (id + \nabla) g). \quad \text{(10)}\]

3) \text{g is unrestricted. Let } h = (T_a^b \xi) \triangleright (T_a^b \xi) g. \text{ Then } (T_a^b \xi) h \text{ is guarded. Using the previous clause we obtain}

\[(h^\dagger)^\dagger = [\eta'', (h^\dagger)^\dagger] \triangleright (T_a^b (id + \nabla) \triangleright h) \quad \text{(Clause 2)}
\]
\[= [\eta'', (h^\dagger)^\dagger] \triangleright (T_a^b (id + \nabla) \triangleright (T_a^b \xi) g) \quad \text{(10)}
\]
\[= [\eta'', (h^\dagger)^\dagger] \triangleright (T_a^b (id + \nabla)(T_a^b \xi) g) \quad \text{(10)}
\]
\[= [\eta'', (h^\dagger)^\dagger] \triangleright (T_a^b (id + \nabla)(T_a^b \xi) g) \quad \text{(10)}
\]
\[= [\eta'', (h^\dagger)^\dagger] \triangleright (T_a^b (id + \nabla) g) \quad \text{(10)}
\]

By (11), we are done once we show that
\[= (g^\dagger)^\dagger. \quad \text{(12)}\]

This will heavily involve nesting of coproduct injections \text{in}, \text{inr}, which we therefore abbreviate to \text{l}, \text{r}, respectively. Moreover, we generally write \text{\triangleright} \text{f} for \text{T_a^b} \xi \triangleright \text{T_a^b} \xi \text{f}. To begin, we note some easily proved unfolding laws for \text{\dagger}, \text{\triangleright}, and \text{\triangleright}:

\[
\text{out} f^\dagger = (T[[\ll, r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] \text{out } f)^\dagger \quad \text{(13)}
\]
\[
\text{out} \triangleright f = (T[[\lll, r], \text{llr}] \text{out } f)^\dagger \quad \text{(14)}
\]
\[
\text{out} \triangleright\triangleright f = (T[[\llll, r], \text{lllr}] \text{out } f)^\dagger. \quad \text{(15)}
\]

It is easy to see that \text{h^\dagger} = (\text{\triangleright} \text{g})^\dagger \text{is guarded. It therefore suffices to show that } (g^\dagger)^\dagger \text{satisfies the unfolding equation for } ((\text{\triangleright} \text{g})^\dagger)^\dagger,

\[= (g^\dagger)^\dagger. \quad \text{(16)}\]

Moreover, note that as a special case of naturality, we generally have

\[= (T(f + \text{id})u)^\dagger. \quad \text{(17)}\]

We proceed to prove (16) by applying out to both sides and unfolding. We start with the left hand side, which by (13) equals

\[(T[[\ll, r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] \text{out}(\text{\triangleright} \text{g})^\dagger)^\dagger. \quad \text{(17)}\]

For readability, we now focus on the subterm \text{out}(\text{\triangleright} \text{g})^\dagger:

\[
\text{out}(\text{\triangleright} \text{g})^\dagger = (T[[\ll, r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] \text{out}(\text{\triangleright} \text{g})^\dagger)^\dagger \quad \text{(by (13))}
\]
\[
= (T[[\ll, r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a](T[[\llll, r], \text{lllr}] \text{out } g)^\dagger)^\dagger \quad \text{(by (15))}
\]
\[
= ((T[[\llll, r], \text{lllr}] \text{out } g)^\dagger)^\dagger \quad \text{(naturality)}
\]
\[
= ((T[[\llll, r], \text{lllr}] \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] \text{out } g)^\dagger)^\dagger \quad \text{(ciodiagonal)}
\]
\[
= (T[[\llll, r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] \text{out } g)^\dagger)^\dagger
\]

Applying naturality, we obtain that the entire left hand side equals

\[
((T[[\llll, r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] + \text{id} T[[\llll, r], r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] \text{out } g)^\dagger)^\dagger
\]
\[
= ((T[[\llll, r], r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] \text{out } g)^\dagger)^\dagger
\]
\[
= (T[[\llll, r], r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] \text{out } g)^\dagger)^\dagger. \quad \text{(ciodiagonal)}
\]

For comparison, we unfold the right hand side using (13) (twice) and naturality, obtaining

\[
\text{out}(g^\dagger)^\dagger = ((T[[\ll, r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] + \text{id} T[[\ll, r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] \text{out } g)^\dagger)^\dagger
\]
\[
= ((T[[\lll, r], \text{lr}([\eta'', (g^\dagger)^\dagger]^\dagger_a] \text{out } g)^\dagger)^\dagger
\]

which by another application of the ciodiagonal law equals the expansion of the left hand side.
It remains to prove (10), equivalently unfolded, i.e. with out applied to both sides. To be precise about bracketing of coproducts, we write \([\text{id}_Y + x, t]\) in place of \(\text{id} + \nabla\). We then have as the unfolding of the left hand side

\[
\text{out} \triangleright (T^n_a [\text{id}_Y + x, t] \triangleright g) = (T[[\text{III}, r], \text{lr}] \text{out} T^n_a [\text{id}_Y + x, t] \triangleright g)^\dagger \tag{by (14)}
\]

\[
= (T[[\text{III}, r], \text{lr}] T([\text{id}_Y + x, t] + (T^n_a [\text{id}_Y + x, t])^b_a) \text{out} \triangleright g)^\dagger \tag{naturality of out}
\]

\[
= (T[[\text{III}, r], \text{lr}] \text{lr}(T^n_a [\text{id}_Y + x, t])^b_a \text{out} \text{lr} g)^\dagger \tag{by (14)}
\]

\[
= ((T[[\text{III}, r], \text{lr}] \text{lr}(T^n_a [\text{id}_Y + x, t])^b_a + \text{id}_X) T[[\text{III}, r], \text{lr}] \text{out} \text{lr} g)^\dagger \tag{naturality}
\]

\[
= ((T[[\text{III}, r], \text{lr}] + \text{lr}(T^n_a [\text{id}_Y + x, t])^b_a \text{out} \text{lr} g)^\dagger \tag{codiagonal}
\]

For comparison, we calculate the unfolding of the right hand side:

\[
\text{out} \triangleright (T^n_a (\text{id} + \nabla) g) = (T[[\text{III}, r], \text{lr}] \text{out} T^n_a [\text{id}_Y + x, t] g)^\dagger \tag{by (14)}
\]

\[
= (T[[\text{III}, r], \text{lr}] T([\text{id}_Y + x, t] + (T^n_a [\text{id}_Y + x, t])^b_a) \text{out} g)^\dagger \tag{naturality of out}
\]

which is thus identical to that of the left hand side.

- **Uniformity.** First, show uniformity under the assumption that \(g\) is guarded. Suppose \(fh = T^n_a (\text{id} + h) g\). It is then sufficient to verify that \(f^\dagger h\) satisfies the unfolding identity for \(g\). Indeed,

\[
f^\dagger h = [\eta^\nu, f^\dagger]^b \text{id} + h) g
\]

\[
= [\eta^\nu, f^\dagger]^b g.
\]

Now consider the general case. Suppose, again we have \(fh = T^n_a (\text{id} + h) g\). We prove the following auxiliary identity:

\[
(\text{out} f)^\dagger h = T((\text{id} + T^n_a (\text{id} + h)) (\text{out} g))^\dagger. \tag{18}
\]

Observe that

\[
(\text{out} f) h = \text{out} T^n_a (\text{id} + h) g
\]

\[
= \pi T((\text{id} + h) + T^n_a (\text{id} + h)) \text{out} g
\]

\[
= T((\text{id} + h) T((\text{id} + T^n_a (\text{id} + h)) + \text{id}) (\text{out} g),
\]

from which by uniformity of the iteration operator of \(T\), we obtain

\[
(\text{out} f) h = T((\text{id} + T^n_a (\text{id} + h)) + \text{id}) (\text{out} g))^\dagger.
\]

After transforming the right hand side by naturality of the iteration operator of \(T\) we arrive at (18).

Next we prove that \((\triangleright f) h = T^n_a (\text{id} + h) \triangleright g\).

\[
(\triangleright f) h = \text{out}^{-1} T((\text{inl} + \text{id}) (\text{out} f)) h
\]

\[
= \text{out}^{-1} T((\text{inl} + \text{id}) T((\text{id} + T^n_a (\text{id} + h)) (\text{out} g))^\dagger
\]

\[
= \text{out}^{-1} T((\text{id} + h) + T^n_a (\text{id} + h)) T((\text{inl} + \text{id}) (\text{out} g))^\dagger
\]

\[
= T^n_a (\text{id} + h) \text{out}^{-1} T((\text{inl} + \text{id}) (\text{out} g))^\dagger \tag{Lemma 22}
\]

\[
= T^n_a (\text{id} + h) \triangleright g. \tag{definition of \triangleright}
\]

As we have shown before, for guarded \(g\) uniformity holds, and therefore \(f^\dagger h = (\triangleright f)^\dagger h = (\triangleright g)^\dagger = g^\dagger\).

- **Compatibility of strength with iteration, i.e. \(\tau^\nu (\text{out} f^\dagger) = ((T^n_a \text{dist}) \tau^\nu (\text{out} f))^\dagger\).** Let \(f\) be guarded with \(\text{out} f = T((\text{inl} + \text{id}) u\). Then, \(f^\dagger = (T \text{dist}) \tau^\nu (\text{out} f)\) is also guarded with \(\text{out} f^\dagger = T((\text{inl} + \text{id}) T((\text{id} + (\tau^\nu)^\dagger)(T \delta) \tau^\nu (\text{out} f))\) where \(\delta\) is as in Lemma 9 (besides guardedness of \(f\), the proof of this equation uses naturality of out and the definitions of \(\tau\) and \(\text{dist}\). The following calculation shows that \(\tau^\nu (\text{out} f^\dagger)\) satisfies the unfolding property for \(((T^n_a \text{dist}) \tau^\nu (\text{out} f))^\dagger\):

\[
\tau^\nu (\text{out} f^\dagger)
\]

24
We show (19) by establishing commutativity of the following diagram where 

\[ Q \]

and hence \( \tau''(\text{id} \times f^\dagger) \) and \( ((T \text{ dist}) \tau''(\text{id} \times f))^{\dagger} \) are equal.

The general case (when \( f \) is not necessarily guarded) reduces to the considered one by means of equation

\[
(T_b^a \text{ dist}) \tau''(\text{id} \times f) = ((T_b^a \text{ dist}) \tau''(\text{id} \times f))^{\dagger},
\]

as follows:

\[
\tau''(\text{id} \times f^\dagger) = \tau''(\text{id} \times (\text{dist} f)^{\dagger}) = \tau''(\text{id} \times (\text{dist} f))^{\dagger} = (\text{dist} (T_b^a \text{ dist}) \tau''(\text{id} \times f))^{\dagger} = (T_b^a \text{ dist} \tau''(\text{id} \times f))^{\dagger}.
\]

We show (19) by establishing commutativity of the following diagram where \( Q = C \times B + C \times A \):

\[
\begin{array}{ccc}
C \times A & \xleftarrow{T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger) = (T_b^a \text{ dist}) \tau''(\text{id} \times f) = ((T_b^a \text{ dist}) \tau''(\text{id} \times f))^{\dagger} = (T_b^a \text{ dist} \tau''(\text{id} \times f))^{\dagger}} & C \times (B + (T_b^a(B + A))_a^b) \\
T(C \times B + (T_b^a(Q)_a^b)) & \xleftarrow{T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger) = \text{dist} (T_b^a \text{ dist}) \tau''(\text{id} \times f) = \text{dist} (T_b^a \text{ dist} \tau''(\text{id} \times f))^{\dagger}} & C \times ((B + A) + (T_b^a(B + A))_a^b) \\
& \xrightarrow{\text{id} \times (\text{dist} f^\dagger)} & \\
& \xrightarrow{T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger) = T_b^a \text{ dist} \tau''(\text{id} \times f)} & T_b^a Q
\end{array}
\]

The bottom triangle commutes as follows:

\[
(T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger) = T_b^a \text{ dist} \text{out}^{\dagger} \text{out} T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger))^{\dagger} = \text{out}^{\dagger} \text{out} T_b^a \text{ dist} \text{out}^{\dagger} T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger) = \text{out}^{\dagger} T_b^a \text{ dist} \text{out}^{\dagger} T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger) = \text{out}^{\dagger} T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger) = T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger).
\]

The middle square commutes by properties of strength and the morphisms \( \text{dist} \) and \( \delta \):

\[
T \text{ dist} + (T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger)) = T \text{ dist} + (T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger)) = T \text{ dist} + (T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger)) = T \text{ dist} + (T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger)) = T \text{ dist} + (T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger)).
\]

This leaves us with the top triangle. Let \( \rho = (\text{id} + (T_b^a \text{ dist} \tau''(\text{id} \times f^\dagger)) \delta \) and let \( \chi \) be such that \( T\chi = \pi \).

\[
(T \rho) \tau(\text{id} \times (\text{out} f)^\dagger) = (T \rho)(T \text{ dist} \tau(\text{id} \times (\text{out} f))^\dagger) = (T(\rho + \text{id})(T \text{ dist} \tau(\text{id} \times (\text{out} f))^\dagger) = (T(\rho + \text{id})(T \text{ dist} \tau(\text{id} \times (\text{out} f))^\dagger).
\]
Finally, we prove (22): 

\[ (T(\rho + \text{id})(T \text{dist})T(\text{id} \times \chi)\tau(\text{id} \times \text{out})(\text{id} \times f))^\dagger \]

with \( g \) completely unguarded, i.e. \( \text{out} g = (T \text{inl})g' \) for some \( g' \). Next we show that

\[ f^\dagger = (h^\dagger g)^\dagger \] (21)

and that

\[ h^\dagger g^\dagger = \triangleright f. \] (22)

In summary, we obtain that \( f^\dagger = (h^\dagger g)^\dagger = (\triangleright f)^\dagger \). The following proofs of (21) and (22) do not depend on the concrete definition of \( \triangleright^1 \) on \( T^0 \), but only use its abstract properties as an iteration operator of a complete Elgot monad and compatibility with the underlying iteration operator for \( T \). Hence, the identity \( f^\dagger = (\triangleright f)^\dagger \) would be valid for any other such operator, but since \( (\triangleright f)^\dagger \) is uniquely defined all of them must be unique.

Let \( g = \text{out}^1(T \text{inl})(\pi \text{ out} f) \), which is, by definition, completely unguarded, and let \( h = \text{out}^1\eta(\text{inl} + \text{id}) \).

Then the proof of (20) runs as follows:

\[
\begin{align*}
[h, \eta' \text{ inr}]^g & = [\text{out}^1\eta(\text{inl} + \text{id}), \eta' \text{ inr}]^g \\
& = [\text{out}^1\eta(\text{inl} + \text{id}), \text{out}^1\eta \text{ inl} \text{ inr}]^g & \text{(Lemma 9)} \\
& = (\text{out}^1\eta)(\text{inl} + \text{id}, \text{inl} \text{ inr})^g \\
& = (\text{out}^1\eta)^g \\
& = \text{out}^1[\eta \chi, \eta \text{ inr}((\text{out}^1\eta \chi)^b)^a] \ast \text{out} g \\
& = \text{out}^1(\eta \chi \ast (\text{out}^1\eta \chi)^b)^a \ast (T \text{inl}) \pi \text{ out} f \\
& = \text{out}^1(\eta \chi)^a \pi \text{ out} f \\
& = \text{out}^1 \pi \text{ out} f \\
& = f.
\end{align*}
\]

where \( \chi = [\text{inl} \text{ inl}, \text{inr} \text{ inr}] \). Equation (21) can be shown as follows:

\[
\begin{align*}
(h^\dagger g)^\dagger & = ((T^0 \text{inl} h, \eta' \text{ inr})^g)^\dagger \\
& = (T^0 \text{id}, \text{inr}[(T^0 \text{inl})h, \eta' \text{ inr}]^g)^\dagger & \text{(naturality)} \\
& = ([h, T^0 \text{id}, \text{inr}][\eta' \text{ inr}]^g)^\dagger \\
& = ([h, \eta' \text{ inr}]^g)^\dagger \\
& = f^\dagger.
\end{align*}
\]

Finally, we prove (22):

\[
\begin{align*}
 h^\dagger g^\dagger & = (\text{out}^1\eta(\text{inl} + \text{id}))^g \\
& = \text{out}^1[\eta(\text{inl} + \text{id}), \eta \text{ inr}(h^\dagger)^b] \ast \text{out} g \\
& = \text{out}^1[\eta(\text{inl} + \text{id}), \eta \text{ inr}(h^\dagger)^b] \ast (T \text{inl})(\pi \text{ out} f) \\
& = \text{out}^1T(\text{inl} + \text{id})(\pi \text{ out} f) \\
& = \triangleright f.
\end{align*}
\]
This finishes the proof.

Lemma 23. Kleisli composition of a complete Elgot monad $T$ can be characterized in terms of iteration as follows:

$$g^* f = [T(inr \ inr) f, T(inl) g]^\dagger \ \text{inl}$$

Proof:

$$[T(inr \ inr) f, T(inl) g]^\dagger \ \text{inl} = [\eta, [T(inr \ inr) f, T(inl) g]^\dagger] \ \text{inr} \ f$$

$$= ([T(inr \ inr) f, T(inl) g]^\dagger \ \text{inr} \ f \ \text{inr}) \ \text{inr} \ f$$

$$= ([T(inr \ inr) f, T(inl) g]^\dagger \ \text{inr} \ f \ \text{inr}) \ \text{inr} \ f$$

$$= g^* f$$

Proof of Theorem 13

The (overlarge) category of complete Elgot monads is formed by (strong) complete Elgot monads and (strong) complete Elgot monad morphisms. The latter are the usual (strong) monad morphisms [26] preserving iteration. Summarized, a complete Elgot monad morphism is a natural transformation $\xi : T \to S$ satisfying the following identities:

$$\xi \eta = \eta \xi f$$

$$\xi f^* = (\xi f)^* \xi$$

$$\xi \tau = \tau (id \times \xi)$$

$$\xi g^* \ ^\dagger = \xi g^\dagger$$

with $f : X \to TY$ and $g : X \to T(Y + X)$.

The proof of the theorem relies on the following sequence of auxiliary lemmas, most crucially on Lemma 14.

Lemma 24. The morphism natural transformation $\text{ext} : T \to T^b_a$ is a complete Elgot monad morphism.

Proof: Let us verify the identities (24) from left to right.

- Compatibility of $\text{ext}$ with unit is a straightforward consequence of Lemma 9: $\text{ext} \eta = \text{out}^1(T\Delta \text{inl}) \eta = \text{out}^1 \eta \ \text{inl} = \eta^\nu$.
- In order to show compatibility of $\text{ext}$ with Kleisli star we call the definition of the latter from Lemma 9:

$$\text{ext}^\dagger g^\dagger \ \text{ext} = (\text{out}^1(T\text{inl}) g)^\dagger \ \text{out}^1(T\text{inl})$$

$$= \text{out}^1(\text{out}^1(T\text{inl}) g \ \eta \ \text{inr}((\text{ext} g)^\dagger)^b) \ ^\dagger (T\text{inl})$$

$$= \text{out}^1((T\text{inl}) g)^*$$

$$= \text{out}^1(T\text{inl}) g^*$$

$$= \text{ext} g^*.$$

- Recall the distributivity transformation $\delta : A \times (B + C^b_a) \to A \times B + (A \times C)^b_a$ from Lemma 9. Then by the corresponding definition of $\tau^\nu$,

$$\tau^\nu (id \times \text{ext}) = \text{out}^1 T(id + (\tau^\nu)^b_a)(T\delta) \tau (id \times \text{out} \ \text{ext})$$

$$= \text{out}^1 T(id + (\tau^\nu)^b_a)(T\delta) \tau (id \times T\text{inl})$$

$$= \text{out}^1 T(id + (\tau^\nu)^b_a)(T\text{inl}) \tau$$

$$= \text{out}^1 (T\text{inl}) \tau$$

$$= \text{ext} \tau.$$

- Since $\text{out}(\text{ext} g) = (T^b_a \text{inl}) g$, then by Theorem 12, $\text{out}(\text{ext} g)^\dagger = (T \text{inl}) g^\dagger$, from which the last identity in (24) follows by composition with $\text{out}^1$ on the left.
Proof of Lemma 14

Let $\xi = \zeta^1$. Suppose $u : a \to Sb$ and $\rho : \mathbb{T} \to \mathbb{S}$ induce $\xi$ as in the statement of the lemma, assume for the time being that $\xi$ is indeed a complete Elgot monads morphism, and let us verify that $\xi_b \text{out}^{-1} \eta \text{inr}(\text{id}, \lambda_\cdot \eta) = u$. Let $w = [\eta \text{inl}, \lambda(x, f), S(\text{inr} f)u(x)]^\ast \rho$. Then

$$\xi_b \text{out}^{-1} \eta \text{inr}(\text{id}, \lambda_\cdot \eta)$$

$$= (w \text{out})^{-1} \eta \text{inr}(\text{id}, \lambda_\cdot \eta)$$

$$= [\eta, (w \text{out})^{-1}]^\ast w \text{out} \text{out}^{-1} \eta \text{inr}(\text{id}, \lambda_\cdot \eta)$$

$$= [\eta, (w \text{out})^{-1}]^\ast [\eta \text{inl}, \lambda(x, f), S(\text{inr} f)u(x)]^\ast \rho \eta \text{inr}(\text{id}, \lambda_\cdot \eta)$$

$$= [\eta, (w \text{out})^{-1}]^\ast [\eta \text{inl}, \lambda(x, f), S(\text{inr} f)u(x)] \text{inr}(\text{id}, \lambda_\cdot \eta)$$

$$= [\eta, (w \text{out})^{-1}]^\ast (\lambda(x, f), S(\text{inr} f)u(x)) \text{out}(\text{id}, \lambda_\cdot \eta)$$

$$= [\eta, \xi_b]^\ast S(\text{inr} \eta)u$$

$$= (\xi_b \eta)^u$$

$$= \eta^u$$

$$= u.$$

Suppose now that $\xi : \mathbb{T}_a^b \to S$ is a morphism of complete Elgot monads. Let $\rho = \xi \text{ext}$ (which is a complete Elgot monad morphism by Lemma 24) and let $u = \xi_b \text{out}^{-1} \eta \text{inr}(\text{id}, \lambda_\cdot \eta)$. Then

$$([\eta \text{inl}, \lambda(x, f), S(\text{inr} f)u(x)]^\ast \rho \text{out})^\dagger$$

$$= ([\eta \text{inl}, \lambda(x, f), S(\text{inr} f)\xi_b \text{out}^{-1} \eta \text{inr}(\eta, \eta)]^\ast \rho \text{out})^\dagger$$

$$= ([\eta \text{inl}, \lambda(x, f), \xi \text{out}^{-1} \text{inl} \eta \text{inr}(\eta, \eta)]^\ast \rho \text{out})^\dagger$$

$$= ([\eta \text{inl}, \lambda(x, f), \xi \text{out}^{-1} \text{inr}(\text{id} \times \eta \text{inr})]^\ast \rho \text{out})^\dagger$$

$$= ([\eta \text{inl}, \lambda(x, f), \xi \text{out}^{-1} \text{inr}(\text{id} \times \eta \text{inr})] \text{ext} \text{out})^\dagger$$

$$= ([\eta \text{inl}, \text{out}^{-1} \eta \text{inr}(\text{id} \times \eta \text{inr})] \text{ext} \text{out})^\dagger$$

$$= \xi([\eta \text{inl}, \text{out}^{-1} \eta \text{inr}(\text{id} \times \eta \text{inr})]^\hat{x} \text{ext} \text{out})^\dagger.$$

To finish the calculation we have to verify that the latter iteration term is equal to the identity. Note that the term under the iteration operator is guarded. Hence, it suffices to show that $\text{id}$ satisfies the corresponding characteristic equation for iteration, i.e. that

$$[\eta, \text{id}]^\hat{x}[\eta \text{inl}, \text{out}^{-1} \eta \text{inr}(\text{id} \times \eta \text{inr})]^\hat{x} \text{ext} \text{out} = \text{id}.$$

Note that we can rephrase the description of Kleisli binding in $\mathbb{T}_a^b$ (Lemma 9) to

$$f \text{out}^{-1} = \text{out}^{-1}[\text{out} f, \eta \text{inr}(\text{out} f)^{\hat{\text{b}}}]^\ast$$

for $f : X \to T^b_\alpha Y$. We have

$$[\eta, \text{id}]^\hat{x}[\eta \text{inl}, \text{out}^{-1} \eta \text{inr}(\text{id} \times \eta \text{inr})]^\hat{x} \text{ext} \text{out}$$

$$= [\eta, [\eta, \text{id}]^\hat{x}[\eta \text{inl}, \text{out}^{-1} \eta \text{inr}(\text{id} \times \eta \text{inr})]^\hat{x} \text{ext} \text{out}$$

$$= [\eta, \text{out}^{-1}[\eta, \text{id}]^\hat{x}[\eta \text{inl}, \text{out}^{-1} \eta \text{inr}(\text{id} \times \eta \text{inr})]^\hat{x} \text{ext} \text{out}$$

$$= [\eta, \text{out}^{-1} \eta \text{inr}(\text{id} \times \text{id})]^\hat{x} \text{ext} \text{out}$$

$$= [\eta, \text{out}^{-1} \eta \text{inr}]^\hat{x} \text{out}^{-1} T \text{inl} \text{out}$$

$$= \text{out}^{-1}[\text{out} [\eta, \text{out}^{-1} \eta \text{inr}], \eta \text{inr} ([\eta, \text{out}^{-1} \eta \text{inr}]^\hat{x} \text{out}^{-1} T \text{inl} \text{out})$$

(by (25))

$$= \text{out}^{-1}[\text{out} [\eta, \text{out}^{-1} \eta \text{inr}], \eta \text{inr} ([\eta, \text{out}^{-1} \eta \text{inr}]^\hat{x} \text{out}^{-1} T \text{inl} \text{out})] = \text{id}.$$
\[ \text{We are left to show that any } \xi_X : T_a^b X \to SX \text{ induced by } u : a \to Sb \text{ and } \rho : T \to \mathbb{S} \text{ is a morphism of complete Elgot monads, that is, } \xi_X \text{ is natural in } X \text{ and satisfies identities (24).} \]

Let us first argue naturality of \( \xi \). Let \( \xi = w^\dagger \). We have

\[
\begin{align*}
wt_a^b f &= [\eta \text{ inl}, \lambda (x, g). \, S \text{ (inr g)} \, u \, (x)] \rho \, \text{ out } T_a^b f \\
&= [\eta \text{ inl}, \lambda (x, g). \, S \text{ (inr g)} \, u \, (x)] \rho T \, (f + (T_a^b f)^b) \, \text{ out} \\
&= [\eta \text{ inl} f, \lambda (x, g). \, S \text{ (inr T}_a^b f g) \, u \, (x)] \rho \, \text{ out} \\
&= S(f + T_a^b f) w
\end{align*}
\]

and thus

\[
\begin{array}{ccc}
T_a^b X & \xrightarrow{w} & S(X + T_a^b X) & \xrightarrow{S(f + \text{id})} & S(Y + T_a^b X) \\
\downarrow T_a^b f & & \downarrow S(f + T_a^b f) & & \downarrow S(\text{id} + T_a^b f) \\
T_a^b Y & \xrightarrow{w} & S(Y + T_a^b Y) & &
\end{array}
\]

commutes. Therefore, the lower triangle in the following diagram commutes by uniformity of the iteration operator:

\[
\begin{array}{ccc}
T_a^b X & \xrightarrow{\xi} & SX \\
\downarrow T_a^b f & & \downarrow S \text{ f} \\
T_a^b Y & \xrightarrow{\xi} & SY
\end{array}
\]

The upper triangle commutes by naturality of the iteration operator:

\[
Sf \xi = (\eta f)^* w^\dagger \\
= ([\text{Sinl} \, \eta f, \eta \text{ inr}^* w]^\dagger \\
= ([\eta \text{ inl} f, \eta \text{ inr}^* w]^\dagger \\
= (S(f + \text{id}) w)^\dagger
\]

The equation \( \xi \eta = \eta \) can be shown as follows:

\[
\begin{align*}
\xi \eta &= [\eta, \xi]^* [\eta \text{ inl}, \lambda (x, f) \, S(\text{inr f}) (u \, (x))]^* \rho \, \text{ out } \eta \\
&= [\eta, \xi]^* [\eta \text{ inl}, \lambda (x, f) \, S(\text{inr f}) (u \, (x))]^* \rho \, \text{ out } \text{ inl} \eta \text{ inl} \\
&= [\eta, \xi]^* \eta \text{ inl} \\
&= \eta.
\end{align*}
\]

Compatibility of \( \xi \) with Kleisli star follows from Lemma 23 and compatibility of \( \xi \) with iteration, which we argue later:

\[
\begin{align*}
\xi g_1 g_2 f &= \xi [T_a^b (\text{inr inr}) f, T_a^b \text{ inl g}]^\dagger \, \text{ inl} \\
&= [S(\text{inr inr}) \xi f, \text{Sinl} \xi g]^\dagger \, \text{ inl} \\
&= (\xi g)^*(\xi f).
\end{align*}
\]
To show compatibility of $\xi$ with strength, consider the following diagram:

$$
\begin{align*}
A \times T_a^b X & \xrightarrow{id \times \rho \text{out}} A \times S(X + (T_a^b X)_a^b) \\
\downarrow\tau & \quad \downarrow S\delta \tau \\
T_a^b(A \times X) & \xrightarrow{\rho \text{out}} S((A \times X) + (A \times T_a^b X)_a^b) \\
\downarrow v\rho \text{out} & \quad \downarrow v \\
S((A \times X) + T_a^b(A \times X)) & \xrightarrow{S(id + \tau)} S((A \times X) + (A \times T_a^b X)),
\end{align*}
$$

where $v = [\eta\text{inl}, \lambda(x, f), S(inr f)u(x)]^*$, i.e. $\xi = (v\rho \text{out})^\dagger$. It is easy to show that this commutes by expanding out $\tau$, using the fact that $\rho$ is a complete monad morphism and observing that

$$
S(id + \tau)v = [\eta\text{inl}, S(id + \tau)\lambda(x, h), S(inr h)(u(x))]^*
$$

$$
= [\eta\text{inl}, \lambda(x, h), S(inr h)(u(x))]^*
$$

$$
= [\eta\text{inl}, \lambda(x, h), S(inr h)(u(x))(id \times \tau^b)]^*
$$

$$
= vS(id + \tau^b).
$$

Thus, by uniformity,

$$
\xi\tau = (v(S\delta)\tau(id \times \rho \text{out}))^\dagger.
$$

On the other hand, by compatibility of strength with iteration, we have

$$
\tau(id \times \xi)
$$

$$
= ((S\text{dist})\tau(id \times v \rho \text{out}))^\dagger
$$

$$
= (S\text{dist})((\tau(id \times v))^\dagger\rho \text{out})
$$

where $v = [\eta\text{inl}, \lambda(x, f), S(inr f)u(x)]$, i.e. $v^* = v$. Therefore, to prove the identity in question, we need to show that the following diagram commutes:

$$
\begin{align*}
S(A \times (X + (T_a^b X)_a^b)) & \xrightarrow{S\delta} S((A \times X) + (A \times T_a^b X)_a^b) \\
\downarrow(\tau(id \times v))^* & \quad \downarrow v \\
S(A \times (X + T_a^b X)) & \xrightarrow{S\text{dist}} S((A \times X) + (A \times T_a^b X)),
\end{align*}
$$

or, differently put, $v\delta = (S\text{dist})\tau(id \times v)$, which, as a morphism out of a coproduct, decomposes into two equations:

On the one hand,

$$
(S\text{dist})\tau(id \times v)(id \times \text{inl})
$$

$$
= (S\text{dist})\tau(id \times \eta\text{inl})
$$

$$
= (S\text{dist})\eta\text{inl}
$$

$$
= \eta\text{inl}
$$

$$
= v\text{inl}
$$

$$
= v\delta(id \times \text{inl}).
$$

On the other hand, after simplifying, we get

$$
((S\text{dist})\tau(id \times v)(id \times \text{inr}))(x, (z, c))
$$

$$
= (S\text{inr})\tau(id \times \lambda(x, f).Sf(u(x)))(x, (z, c))
$$

$$
= S\text{inr}(\tau(x, (Sc)(u(z))))
$$

as well as

$$
\text{v}\delta(x, \text{inr}(z, c))
$$

$$
= \text{v}(\text{inr}(z, \lambda v.(x, c(v))))
$$
\[ S(\text{inr}(\lambda v.(x, c(v))))(u(z)) \]

(for the first step, recall the explicit lambda-expression for \( \delta \)). Identity of these expressions follows by the fact that we are working in a bicartesian closed base category, which allows us to give an explicit lambda-expression for the strength, namely \( \tau = \lambda(a, b).S(\lambda c.(a, c))(b) \).

Finally, we are left to show that

\[ \xi f^\dagger = (\xi f)^\dagger \]  \hspace{1cm} (26)

for any \( f : X \to T_a^b(Y + X) \). For the sake of brevity let us denote \( \lambda(x, f). (Sf)(u(x)) \) by \( ev_u \). Then \( \xi = ([\eta \text{inl}, S(\text{inr}) ev_u]^* \rho \text{out})^\dagger \).

First, we argue that w.l.o.g. \( \xi \) may be taken to be guarded. Assuming that \( \xi(\triangleright f)^\dagger = (\xi \triangleright f)^\dagger \), since by definition \( \xi(\triangleright f)^\dagger = \xi f^\dagger \), we can deduce (26) from the equality \( (\xi \triangleright f)^\dagger = (\xi f)^\dagger \). To show the latter, consider the morphism \( w \) given by the composition

\[
X \xrightarrow{\text{out} f} T((Y + X) + T_a^b(Y + X)) \xrightarrow{[\eta \text{inl} \text{inr}, (S \text{inl}) \xi^* ev_u]^* \rho} S((Y + X) + X).
\]

Now, on the one hand

\[
(S[\text{id}, \text{inr}]w)^\dagger = ([\eta \text{inl}, \text{inr}, \xi^* ev_u]^* \rho \text{out} f)^\dagger \\
= ([\eta, \xi^* ev_u]^* \rho \text{out} f)^\dagger \\
= ([\eta, \xi^* ev_u]^* \rho \text{out} f)^\dagger \\
= (\xi f)^\dagger
\]

and on the other hand, by naturality of \( \_^\dagger \),

\[
(w^\dagger)^\dagger = (([\eta \text{inl}, \eta \text{inr}, S \text{inl} \xi^* ev_u]^* \rho \text{out} f)^\dagger)^\dagger \\
= (([S \text{inl}]([\eta \text{inl}, \xi^* ev_u], \eta \text{inr})*([\eta \text{inl} \text{inr}, \eta \text{inr}] \eta \text{inr} \text{inr})* \rho \text{out} f)^\dagger)^\dagger \\
= ([\eta, \xi^* ev_u]^* \rho \text{out} f)^\dagger \\
= ([\eta, \xi^* ev_u]^* \rho \text{out} f)^\dagger \\
= ([\eta, \xi^* ev_u]^* \rho \text{out} f)^\dagger \\
= ([\eta, \xi^* ev_u]^* \rho \text{out} f)^\dagger \\
= (\xi f)^\dagger.
\]

Therefore, we obtain the equality of \( (\xi f)^\dagger \) and \( (\xi \triangleright f)^\dagger \) by the codiagonal property of \( \_^\dagger \). We thus proceed under the assumption that \( f \) is guarded, i.e. \( \text{out} f = T(\text{inl} + \text{id})g \) for some \( g : X \to T(Y + T_a^b(Y + X)) \).

We introduce the following morphism \( w \),

\[
T_a^b(Y + X) \xrightarrow{\rho[\text{inl}, \text{inr}, (T \text{inl} \text{inr}) ev_u]^* g, (T \text{inr} ev_u)^*]} S((Y + T_a^b(Y + X)) + T_a^b(Y + X)).
\]

Then, on the one hand, using dinaturality,

\[
(w^\dagger)^\dagger = ([\rho[\text{inl}, \text{inr}, (T \text{inl} \text{inr}) ev_u]^* g, (T \text{inr} ev_u)^* out])^\dagger \\
= ([\eta \text{inl}, (S \text{inr}) ev_u]^* \rho g)^* ([\eta \text{inl}, (S \text{inr} ev_u]^* \rho \text{out})^\dagger \\
= ([\eta \text{inl}, (S \text{inr} ev_u]^* \rho g]^* \xi^\dagger \\
= [\eta, ([\eta \text{inl}, \xi]^* [\eta \text{inl} \text{inr}, (S \text{inr} ev_u]^* \rho g)^\dagger]^* \xi \\
= [\eta, ([\eta \text{inl}, \xi]^* [\eta \text{inl} \text{inr}, (S \text{inr} ev_u]^* \rho g)^\dagger]^* \xi \\
= [\eta, ([\eta \text{inl}, \xi]^* [\eta \text{inl} \text{inr}, (S \text{inr} ev_u]^* \rho f)^\dagger]^* \xi \\
= [\eta, (\xi f)^\dagger]^* \xi
\]

31
and hence \((w^\dagger)[\eta' \lbrace \text{inr} \rbrace] = (\xi f)^\dagger\). Let us furthermore introduce the following morphism \(t\):

\[
T^h_a(Y + X) \xrightarrow{\text{out}} T((Y + X) + T^h_a(Y + X)^\dagger_a)
\]

\[
\xrightarrow{[\text{inl inl}, \text{inl}]} T(Y + T^h_a(Y + X)^\dagger_a)
\]

\[
\xrightarrow{T(\text{inl} + (\eta' \lbrace \text{inr} \rbrace))} T((Y + T^h_a(Y + X)) + T^h_a(Y + T^h_a(Y + X)^\dagger_a))
\]

\[
\xrightarrow{\text{out}^\dagger} T^h_a(Y + T^h_a(Y + X)).
\]

Recall that guardedness of \(f\) means that out \(f\) factors through \(X \rightarrow T(Y + T^h_a(Y + X)^\dagger_a)\). Observe that \(t\) satisfies a stronger assumption. Let us call \(f\) strongly guarded if there is \(h : X \rightarrow T(Y + X^\dagger_a)\) such that out \(f = T(\text{inl} + (\eta' \lbrace \text{inr} \rbrace))h\). The morphism \(t\) is strongly guarded by definition. Furthermore,

\[
[\eta, \xi]^*[\eta \lbrace \text{inl}, (\text{inr}) \text{ ev}_u \rbrace]^* \rho [\eta \lbrace \text{inl} \text{ ev}_u, T((\eta' \lbrace \text{inr} \rbrace)^\dagger_a)g], \eta \lbrace \text{inr} \rbrace]^* \text{ out}
\]

\[
= [\eta, \xi]^*[\eta \lbrace \text{inl}, (\text{inr}) \text{ ev}_u \rbrace]^*[\eta \lbrace \text{inl} \text{ ev}_u, S(\text{inl} + (\eta' \lbrace \text{inr} \rbrace)^\dagger_a)g], \eta \lbrace \text{inr} \rbrace]^* \text{ out}
\]

\[
= [\eta, \xi]^*[\eta \lbrace \text{inl} \text{ ev}_u, S(\text{inl} \eta \lbrace \text{inr} \rbrace) \text{ ev}_u \rbrace]^* \rho \text{ out}
\]

\[
= [\eta, \xi]^*[\eta \lbrace \text{inl}, \text{inl}, (\text{inr}) \text{ ev}_u \rbrace]^*[\eta \lbrace \text{inl} \text{ ev}_u, S(\text{inl} \eta \lbrace \text{inr} \rbrace) \text{ ev}_u \rbrace]^* \rho \text{ out}
\]

\[
= [\eta, \xi]^*[\eta \lbrace \text{inl}, \text{inl}, \text{inr} \text{ ev}_u \rbrace]^*[\eta \lbrace \text{inl} \text{ ev}_u, S(\text{inl} \eta \lbrace \text{inr} \rbrace) \text{ ev}_u \rbrace]^* \rho \text{ out}
\]

\[
= [\eta, \xi]^*[\eta \lbrace \text{inl}, \text{inl}, \text{inr} \text{ ev}_u \rbrace]^*[\eta \lbrace \text{inl} \text{ ev}_u, S(\text{inl} \eta \lbrace \text{inr} \rbrace) \text{ ev}_u \rbrace]^* \rho \text{ out}
\]

\[
= [\eta, \xi]^*[\eta \lbrace \text{inl}, \text{inl}, \text{inr} \text{ ev}_u \rbrace]^*[\eta \lbrace \text{inl} \text{ ev}_u, S(\text{inl} \eta \lbrace \text{inr} \rbrace) \text{ ev}_u \rbrace]^* \rho \text{ out}
\]

Let us assume for the moment that \(\xi t^\dagger = (\xi t)^\dagger\) for strongly guarded \(t\). Then, by the above calculations, \((\xi f)^\dagger = (w^\dagger)[\eta' \lbrace \text{inr} \rbrace] = (S[\text{id}, \text{inr}]w)^\dagger[\eta' \lbrace \text{inr} \rbrace] = (\xi t)^\dagger[\eta' \lbrace \text{inr} \rbrace].\) In order to show that the right-hand side is equal to \(\xi f^\dagger\) we prove that \(t^\dagger = [\eta'', f^\dagger]^\dagger\). for then \((\xi t^\dagger)[\eta' \lbrace \text{inr} \rbrace] = [\xi f^\dagger][\eta' \lbrace \text{inr} \rbrace] = \eta^\dagger[\eta' \lbrace \text{inr} \rbrace].\) Since \(t\) is guarded, it suffices to show that \([\eta'', f^\dagger]^\dagger\) satisfies the unfolding law for \(t^\dagger\). It is easy to verify that out \(f^\dagger = T((\text{id} + (\eta'' \lbrace \text{inr} \rbrace)^\dagger_a)g).\) Then we have

\[
\text{out}[\eta'', f^\dagger]^\dagger = [\text{out}[\eta'', f^\dagger], \eta \lbrace \text{inr} \rbrace]^* \text{ out}
\]

\[
= [[\eta \lbrace \text{inl}, \text{out}^\dagger f^\dagger], \eta \lbrace \text{inr} \rbrace]^* \text{ out}
\]

\[
= [[\eta \lbrace \text{inl}, T((\text{id} + (\eta'' \lbrace \text{inr} \rbrace)^\dagger_a)g], \eta \lbrace \text{inr} \rbrace]^* \text{ out}
\]

while, on the other hand,

\[
\text{out}^\dagger f^\dagger = [\text{out}^\dagger f^\dagger, \eta \lbrace \text{inr} \rbrace]^* \text{ out}
\]

\[
= \text{out}^\dagger f^\dagger \eta \lbrace \text{inr} \rbrace]^* \text{ out}
\]

\[
= \text{out}^\dagger f^\dagger T((\text{id} + (t^\dagger)^\dagger_a)g], \eta \lbrace \text{inr} \rbrace]^* \text{ out}
\]

Hence, indeed, \([\eta'', f^\dagger]^\dagger = t^\dagger\).

Finally, let us show (26) with strongly guarded \(f\). Suppose, \(h\) is such that out \(f = T(\text{inl} + (\eta' \lbrace \text{inr} \rbrace)^\dagger_a)h.\) Recall that \(\xi = (\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^* \text{ out}^\dagger\). By uniformity, it suffices to show that

\[
[\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^* \rho \text{ out} f^\dagger = S(\text{id} + f^\dagger)\xi f.
\]

On the one hand,

\[
[\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^* \rho \text{ out} f^\dagger
\]

\[
= [\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^* \rho \text{ out}^\dagger f^\dagger f
\]

\[
= [\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^*[\text{out}[\eta'', f^\dagger], \eta \lbrace \text{inr} \rbrace]^* \rho \text{ out}
\]

\[
= [\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^*[\text{out}^\dagger f^\dagger, \eta \lbrace \text{inr} \rbrace]^* \rho \text{ out}
\]

\[
= [\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^*[\text{out}[\eta'', f^\dagger], \eta \lbrace \text{inr} \rbrace]^* \rho \text{ out}
\]

\[
= [\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^*[\text{out}^\dagger f^\dagger, \eta \lbrace \text{inr} \rbrace]^* \rho \text{ out}
\]

\[
= [\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^*[\text{out}[\eta'', f^\dagger], \eta \lbrace \text{inr} \rbrace]^* \rho \text{ out}
\]

\[
= [\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^*[\text{out}^\dagger f^\dagger, \eta \lbrace \text{inr} \rbrace]^* \rho \text{ out}
\]

\[
= [\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^*[\text{out}[\eta'', f^\dagger], \eta \lbrace \text{inr} \rbrace]^* \rho \text{ out}
\]

\[
= [\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^*[\text{out}^\dagger f^\dagger, \eta \lbrace \text{inr} \rbrace]^* \rho \text{ out}
\]

And on the other hand,

\[
S(\text{id} + f^\dagger)\xi f = S(\text{id} + f^\dagger)[\eta', \xi]^*[\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^* \rho \text{ out}
\]

\[
= S(\text{id} + f^\dagger)[\eta', \xi]^*[\eta \lbrace \text{inl}, S(\text{inr}) \text{ ev}_u \rbrace]^* \rho T(\text{inl} + (\eta'' \lbrace \text{inr} \rbrace)^\dagger_a)h
\]

32
By Lemma 14 (with corresponding morphism for the right hand-side to prove that it is also a monad morphism). We have to prove that there is a unique monad morphism \( \xi \) induced by \( R \) which exactly means the \( \xi \) takes \( \beta \) to \( \alpha \). On the other hand, any other morphism \( \theta : L^b_a \to S \) for which \( u \) decomposes as above with \( \xi \) replaced by \( \theta \), corresponds to \( u \) under the bijection of Lemma 14, and hence such \( \theta \) is identically \( \xi \).

By Lemma 14 (with \( S = T^b_a, T = L^b_a \)), the Kleisli morphism

\[
a \xrightarrow{\text{out}^{-1} \text{inl} \eta \lambda} L^b_a \xrightarrow{\xi} S_b,
\]

which exactly means the \( \xi \) takes \( \beta \) to \( \alpha \). On the other hand, any other morphism \( \theta : L^b_a \to S \) for which \( u \) decomposes as above with \( \xi \) replaced by \( \theta \), corresponds to \( u \) under the bijection of Lemma 14, and hence such \( \theta \) is identically \( \xi \).

By Lemma 14 (with \( S = T^b_a, T = L^b_a \)), the Kleisli morphism

\[
a \xrightarrow{\text{out}^{-1} \text{inl} \eta \lambda} L^b_a \xrightarrow{\xi} S_b
\]

induces a monad morphism \( \xi : L^b_a \to T^b_a \). We next show that \( T^b_a \) is the coproduct of \( T \) and \( L^b_a \) with \( \xi \) and \( \text{ext} : T \to T^b_a \) being coproduct injections. Let \( \mathbb{R} \) be a complete Elgot monad and let \( \rho : T \to \mathbb{R}, \theta : L^b_a \to \mathbb{R} \) be two complete Elgot monad morphisms. We have to prove that there is a unique \( \kappa : T^b_a \to \mathbb{R} \) such that

\[
\rho = \kappa \text{ ext} \quad \theta = \kappa \xi. \quad (27)
\]

By Lemma 14, there is a Kleisli morphism \( u : a \to Rb \) induced by \( \theta \). Again, by Lemma 14, the pair \( u, \rho \) induces a monad morphism \( T^b_a \to \mathbb{R} \) which we take as \( \kappa \). Let us show the left part of (27):

\[
\kappa \text{ ext} = ([\eta \text{ inl}, \lambda(x, f) \cdot \text{S(inr} f) u(x)] \text{ out}^{-1} T \text{ inl})
\]

\[
= [\eta, \kappa]^* [\eta \text{ inl}, \lambda(x, f) \cdot \text{S(inr} f) u(x)] \text{ out}^{-1} T \text{ inl}
\]

\[
= [\eta, \kappa]^* [\eta \text{ inl}, \lambda(x, f) \cdot \text{S(inr} f) u(x)] \text{ out}^{-1} T \text{ inl}
\]

\[
= [\eta, \kappa]^* (R \text{ inl}) \rho
\]

\[
= \rho.
\]

In order to show the right-hand side of (27), observe that by Lemma 14 both side of the equation in question are completely identified by the corresponding Kleisli morphism \( a \to Rb \). For \( \rho \), such morphism is by definition \( u \). Let us calculate the corresponding morphism for the right hand-side to prove that it is also \( u \):

\[
\kappa \xi = \kappa \text{ ext} = \kappa \text{ inl} \eta \lambda \text{ inl} \eta \lambda \text{ inl} \eta \lambda = u.
\]

Finally, we show that \( \kappa \) satisfying (27) is unique. Suppose, \( \kappa' \) is another such. By Lemma 14, \( \kappa' \) induces \( u' : a \to Rb \) and \( \rho' : T \to \mathbb{R} \). We will be done once we show that \( u = u' \) and \( \rho = \rho' \). On the one hand, by definition, \( \rho' = \kappa' \text{ ext} = \rho \). One the other hand,

\[
u = \theta_b \text{ out}^{-1} \eta \text{ inl} \lambda \text{ inl} \eta \lambda
\]

\[
= \kappa' \xi_b \text{ out}^{-1} \eta \text{ inl} \lambda \text{ inl} \eta \lambda
\]

\[
= \kappa \text{ out}^{-1} \eta \text{ inl} \lambda \text{ inl} \eta \lambda = u
\]

and thus we are done.

This finishes the proof of Lemma 14. We proceed with the proof of Theorem 13.

- The fact that \( L^b_a \) is a complete Elgot monad follows from the assumption and Theorem 12. We have to show that for any complete Elgot monad \( S \) equipped with an algebraic operation \( \alpha : S^b \to S^a \) there is a unique monad morphism \( \xi : L^b_a \to S \) compatible with the corresponding algebraic operation \( \beta : (L^b_a)^b \to (L^b_a)^a \), i.e. \( \xi \alpha = \alpha \xi^b \) where \( \beta_X(f : b \to L^b_a X)(x : A) = \text{out}^{-1} \text{inl}(x, f) \).

Recall that algebraic operations dualy correspond to generic effects [32], i.e. \( \alpha \) induces a Kleisli morphism \( u : a \to Sb \).

By Lemma 14, \( u \) induces a monad morphism \( \xi : L^b_a \to S \). According to Lemma 14, \( u : a \to Sb \) is now representable as the composition

\[
a \xrightarrow{\text{out}^{-1} \text{inl} \eta \lambda} L^b_a \xrightarrow{\xi} S_b
\]

which exactly means the \( \xi \) takes \( \beta \) to \( \alpha \). On the other hand, any other morphism \( \theta : L^b_a \to S \) for which \( u \) decomposes as above with \( \xi \) replaced by \( \theta \), corresponds to \( u \) under the bijection of Lemma 14, and hence such \( \theta \) is identically \( \xi \).
Lemma 14 shows that a morphism generated by a generic effect is a complete Elgot monad morphism. In fact, this part of the claim can be generalised.

**Lemma 25.** Let \( T \) and \( S \) be complete Elgot monads, let \( \rho \) be a complete Elgot monad morphism and let \( h : a \to T^b a \). Then the following morphism \( \xi \)

\[
T^b X \xrightarrow{\text{out}} T(X + a \times (T^b a) \times) \xrightarrow{S(id + \lambda(x,f) \cdot f^b h(x)) \rho} S(X + T^b a \times)
\]

induces a complete Elgot monad morphism \( \xi^\dagger \).

**Proof:** Let us show that \( \xi^\dagger \) is a monad morphism first. Foremost we have

\[
\begin{align*}
\xi^\dagger \eta'' &= [\eta, \xi^\dagger]^* S(id + \lambda(x,f) \cdot f^b h(x)) \rho \text{ out} \eta'' \\
&= [\eta, \xi^\dagger]^* S(id + \lambda(x,f) \cdot f^b h(x)) \rho \eta \text{ inl} \\
&= \eta.
\end{align*}
\]

Then we show the identity

\[
\xi^\dagger \xi = (\xi^\dagger t)^* \xi^\dagger
\]

in the two partial cases \( t = \eta'' u \) and \( t = \text{id}^b \), which jointly imply the general case as follows:

\[
\begin{align*}
\xi^\dagger \xi &= (\xi^\dagger)^* \xi^\dagger \\
&= (\xi^\dagger)^* (\xi^\dagger \eta'' t)^* \\
&= ((\xi^\dagger)^* (\xi^\dagger \eta'' t)^* \xi^\dagger \\
&= (\xi^\dagger t)^* \xi^\dagger.
\end{align*}
\]

Note that \( (\eta'' u)^\dagger = T^b a \) and proceed with the following identities:

\[
\begin{align*}
\xi^\dagger (T^b a) &= (Su)^\dagger \\
\xi^\dagger \mu &= (\xi^\dagger)^b (\xi^\dagger)^\dagger
\end{align*}
\]

where we denote \( \mu = \text{id}^b \) (indeed \( \mu \) is the multiplication transformation of \( T^b a \)).

In order to show (29), observe that

\[
\begin{align*}
\xi(T^b a) &= S(id + \lambda(x,f) \cdot f^b h(x)) \rho \text{ out} (T^b a) \\
&= S(id + \lambda(x,f) \cdot f^b h(x)) T(u + (T^b a)^\dagger) \rho \text{ out} \\
&= S(u + \lambda(x,f) \cdot ((T^b a)^f b h(x)) \rho \text{ out} \\
&= S(id + (T^b a) S(u + \lambda(x,f) \cdot f^b h(x)) \rho \text{ out}.
\end{align*}
\]

By uniformity and by naturality, we obtain

\[
\begin{align*}
\xi^\dagger (T^b a) &= (Su)(S(id + \lambda(x,f) \cdot f^b h(x)) \rho \text{ out})^\dagger \\
&= (Su)(S(id + \lambda(x,f) \cdot f^b h(x)) \rho \text{ out})^\dagger \\
&= (Su)^\dagger.
\end{align*}
\]

Let us proceed with the proof of (30). Observe the following

\[
\begin{align*}
\xi \mu &= S(id + \lambda(x,f) \cdot f^b h(x)) \rho \text{ out} \mu \\
&= S(id + \lambda(x,f) \cdot f^b h(x)) \rho [\text{ out, } \eta'' \text{ inr } \mu]^b ]^* \text{ out} \\
&= S(id + \lambda(x,f) \cdot f^b h(x)) [\rho \text{ out, } \eta'' \text{ inl } \mu]^* \text{ out} \\
&= [\xi, \lambda(x,f) \cdot f^b h(x)] \rho \text{ out} \\
&= S(id + \mu) [S(id + \eta'' \xi) \lambda(x,f) \cdot \eta \text{ inr } f^b h(x)]^* \text{ out}
\end{align*}
\]

and therefore, by uniformity and by coinductive,

\[
\begin{align*}
\xi^\dagger \mu &= ([S(id + \eta'' \xi) \lambda(x,f) \cdot \eta \text{ inr } f^b h(x)]^* \text{ out})^\dagger \\
&= (([S(id + \eta'' \xi) \lambda(x,f) \cdot \eta \text{ inr } f^b h(x)]^* \rho \text{ out})^\dagger
\end{align*}
\]

34
Finally, the latter can be transformed as follows:

\[
(S(id + \eta^\nu)\xi^\dagger) = [\eta, (S(id + \eta^\nu)\xi^\dagger)S(id + \eta^\nu)] = [\eta, (\xi^\dagger\eta^\nu)(\xi^\dagger)] = [\eta, \xi^\dagger(\xi^\dagger\eta^\nu)] = [\eta, \xi^\dagger]\quad(\dagger)\]

which completes the proof of (30). At step (\dagger) we applied the identity \((S(id + \eta^\nu)\xi^\dagger)\) obtained by applying uniformity to the following trivial equation:

\[
S(id + \eta^\nu)\xi^\dagger\eta^\nu = (\xi^\dagger\eta^\nu)\]

Compare the following pair of morphisms:

\[
\xi = [\eta \text{ inl}, \lambda(x, f) \cdot \eta\text{ inr}\ f^g h(x)]^\dagger
\]

\[
\zeta = [\eta \text{ inl}, \lambda(x, f) \cdot S(\text{inr}\ f) \xi^\dagger h(x)]^\dagger
\]

The morphism \(\zeta^\dagger\) is exactly of the form figuring in Lemma 14 and therefore is a complete Elgot monad morphism. Hence, we will be done once we show that \(\xi^\dagger = \zeta^\dagger\). Let us rewrite \(\zeta^\dagger\) as follows:

\[
\zeta^\dagger = ([\eta \text{ inl}, \lambda(x, f) \cdot S(\text{inr}\ f))^\dagger h(x)]^\dagger\rho\text{ out})
\]

\[
= [\eta, ([\eta \text{ inl}, \rho\text{ out}] \lambda(x, f) \cdot S(\text{inr}\ f)^\dagger h(x)]^\dagger\rho\text{ out}
\]

\[
= [\eta, \lambda(x, f) \cdot (\text{out}\ f)^\dagger h(x)]^\dagger\rho\text{ out}
\]

\[
= [\eta, \lambda(x, f) \cdot \xi^\dagger(\text{out}\ f)^\dagger h(x)]^\dagger\rho\text{ out}
\]

(28)

At the latter step we used (28) and the easily verifiable equation \(\xi^\dagger\text{ ext} = \rho\). Our further goal is to show that

\[
(\lambda(x, f), \xi^\dagger(\text{out}\ f)^\dagger h(x))^\dagger = \lambda(x, f), \xi^\dagger f^g h(x)
\]

(31)

which will allow us to complete the proof as follows:

\[
\zeta^\dagger = [\eta, \lambda(x, f) \cdot f^g h(x)]^\dagger\rho\text{ out}
\]

\[
= [\eta, \xi^\dagger]^\dagger f h(x)
\]

\[
= \xi^\dagger
\]

Let \(t = \lambda(x, f) \cdot (\text{out}\ f)^\dagger h(x)\) and note the following:

\[
(S[\text{id} + t, \text{inr}]\xi)^\dagger t = (S[\text{id} + t]S((\text{id} + t) + \text{id})\xi)^\dagger t
\]

\[
= (S((\text{id} + t) + \text{id})\xi)^\dagger t
\]

\[
= (S[\text{id} + t]\xi)^\dagger t
\]

(32)

Therefore, (31) can be reformulated as follows:

\[
(S[\text{id} + t, \text{inr}]\xi)^\dagger t = \lambda(x, f) \cdot f^g h(x).
\]

We simplify the left-hand side of (32) by showing that \((S[\text{id} + t, \text{inr}]\xi)^\dagger = \xi^\dagger(\text{out}^{-1}\eta)^\dagger\) by uniformity from

\[
S[\text{id} + (\text{out}^{-1}\eta)^\dagger S[\text{id} + t, \text{inr}]\xi = \xi(\text{out}^{-1}\eta)^\dagger
\]

whose proof is as follows. First observe that

\[
(\text{out}^{-1}\eta)^\dagger(\text{out}\ f)^\dagger
\]

\[
= (\text{out}^{-1}\eta)^\dagger\text{ out}^{-1} T\text{ inl}\text{ out}\ f)^\dagger
\]

\[
= (\text{out}^{-1}\eta), \eta\text{ inr}(\text{out}^{-1}\eta)^\dagger(\text{out}\ f)^\dagger
\]

\[
= (\text{out}^{-1}\eta)^\dagger(\text{out}\ f)^\dagger
\]

35
\[ = f^\delta. \]

Then, on the one hand, we have
\[
S(id + (\eta^1)^b) S[id + t, \text{inr}] = S[id + (\eta^1)^b t, \text{inr}(\eta^1)^b \xi] = S[id + (\eta^1)^b t, \lambda(x, f), \text{inr}(\eta^1)^b f^\delta h(x)] \rho \text{out}
\]
\[
= S[id + \lambda(x, f), (\eta^1)^b (\text{ext out})^f h(x), \lambda(x, f), \text{inr}(\eta^1)^b f^\delta h(x)] \rho \text{out}
\]
\[
= S[id + \lambda(x, f), f^\delta h(x), \lambda(x, f), \text{inr}(\eta^1)^b f^\delta h(x)] \rho \text{out},
\]
while on the other hand
\[
\xi(\eta^1)^t = S(id + \lambda(x, f), f^\delta h(x)) \rho [\eta, \eta \text{inr}((\eta^1)^b)^* \text{out}]
\]
\[
= S(id + \lambda(x, f), f^\delta h(x)) S[id, \text{inr}((\eta^1)^b)^* \text{out}]
\]
\[
= S[id + \lambda(x, f), f^\delta h(x), \lambda(x, f), \text{inr}((\eta^1)^b)^* f^\delta h(x)] \rho \text{out}
\]
\[
= S[id + \lambda(x, f), f^\delta h(x), \lambda(x, f), \text{inr}(\eta^1)^b f^\delta h(x)] \rho \text{out}.
\]

We thus reduce to (32)
\[
\xi(\eta^1)^t = \lambda(x, f), \xi^1 f^\delta h(x).
\]
The latter equation follows trivially:
\[
\xi(\eta^1)^t = \lambda(x, f), \xi^1 (\eta^1)^b (\text{ext out})^f h(x)
\]
\[
= \lambda(x, f), \xi^1 f^\delta h(x).
\]
The proof of the lemma is thus completed.

---

**Proof of Theorem 15**

We begin by showing that \( L \) is an \( \omega \)-continuous monad; by Theorem 7 it will follow that \( L \) is a complete Elgot monad.

The base category \( C \) is, a fortiori, extensive; in any extensive category, \( L \) is the partial map classifier for partial morphisms whose domains are coproduct injections. Thus, the Kleisli category of \( L \) inherits orderings on its hom-sets from the extension ordering on partial functions; the fact that coproduct injections are closed under unions in \( C \) guarantees that these orderings are \( \omega \)-complete (note that any ascending chain of coproduct injections qua subobjects can, using universality of coproducts, be transformed into a disjoint union of coproduct injections). Using the properties of hyper-extensive categories, one can show that this induces a \( \omega \)\( \text{Cppo} \)-enrichment of \( C \L \) that satisfies all additional conditions imposed in Definition 4.

Note that any complete Elgot monad \( T \) for any \( X \in \{C\} \) possesses a global element \( \bot_X = \delta_X^1 \colon 1 \to TX \) where \( \delta_X = \eta \text{inr} : 1 \to T(X + 1) \). It follows by naturality of iteration that \( \bot_X \) is actually natural in \( X \). Moreover, \( \bot \) is preserved by complete Elgot monad morphisms. It is easy to see that \( \xi_X = [(\eta, \bot_X)] \) yields a complete Elgot monad morphism \( \xi : L \to T \). On the other hand it is the only such because for any other complete Elgot monad morphism \( \theta : L \to T \) one would have \( \theta \text{inl} = \theta \eta = \eta = \xi \text{inl} \) and \( \theta \text{inr} = \theta \bot = \bot = \xi \text{inr} \) implying \( \theta = \xi \).

**Proof of Equation (*)**

Let us denote case \( z \) of \( \text{inl} y \mapsto \psi(y); \text{inr} x \mapsto (\text{do} z \leftarrow q; f(z)) \) by \( r \) for the sake of brevity and let
\[
\gamma = [\Delta ; \Delta, f : B + A \to B; \Gamma, \pi : B + A, \nu : B \to [B]_\Delta, f \vdash_c r : B],
\]
\[
\gamma_1 = [\Delta ; \Delta, \pi : B + A]
\]
\[
\gamma_2 = [\Delta ; \Delta, \pi : B + A]
\]
where \( \Delta \) is the joint effect context of \( p \) and \( q \) not containing \( f : B + A \to B \). By definition we have
\[
\gamma(l, z, u) = [u, \lambda x.((\eta^1)^A \text{inr}(\text{id}, \lambda_\eta^{A+f}))^* \text{ext} \gamma_2(l, x)](z).
\]

Then
\[
[\Delta ; \Gamma \vdash_c \text{iter inr} x \leftarrow p; q : B](l) = [\eta^A, \lambda x. \gamma_2(l, x)]^* (\gamma_1(l))
\] (33)
Now, in order to show the identity of (33) and (34) it suffices to obtain

\[ \psi(l, w) = T_\Delta (pr_2 + \gamma)(T_\Delta \text{dist})\tau^\Delta(l, \text{out}(w)) = T_\Delta (\text{id} + \lambda(z, u). \gamma(l, z, u))(\text{out}(w)). \]

Let us further rewrite the right-hand side of (34):

\[
\psi^+(l, (\text{out}^1 \eta^\Delta \text{inr}(\text{id}, \lambda_\text{inl} \eta^\Delta, f))^* (\text{ext} \gamma_1(l))) = \\
[\eta^\Delta, \lambda w. \psi^+(l, w)]^* \psi(l, (\text{out}^1 \eta^\Delta \text{inr}(\text{id}, \lambda_\text{inl} \eta^\Delta, f))^* (\text{ext} \gamma_1(l))) = \\
\{[(\eta^\Delta, \lambda w. \psi^+(l, w))^* T_\Delta (\text{id} + \lambda(z, u). \gamma(l, z, u)) \text{out}(\text{out}^1 \eta^\Delta \text{inr}(\text{id}, \lambda_\text{inl} \eta^\Delta, f))^* (\text{ext} \gamma_1(l)) \} = \\
\{[(\eta^\Delta, \lambda w. \psi^+(l, w))^* T_\Delta (\lambda z. \gamma(l, z, \eta^\Delta, f))^* (\text{ext} \gamma_1(l)) \} = \\
\{(\lambda z. \psi^+(l, \gamma(l, z, \eta^\Delta, f)))^* (\text{ext} \gamma_1(l)) \}\]

Now, in order to show the identity of (33) and (34) it suffices to obtain

\[ [\eta^\Delta, \lambda x. \gamma_2^\Delta(l, x)] = \lambda z. \psi^+(l, \gamma(l, z, \eta^\Delta, f)). \]

Note that

\[ T_\Delta (\text{id} + \text{out})\psi(l, w) = T_\Delta (\text{id} + \lambda(z, u). \text{out} \gamma(l, z, u))(\text{out}(w)) \]

and therefore, by uniformity and by Lemma 28,

\[ \psi^+(l, w) = (T_\Delta (\text{id} + \lambda(z, u). \text{out} \gamma(l, z, u)))^t(\text{out}(w)) = [\eta^\Delta, \lambda(z, u). (\text{out} \gamma)^t(l, z, u)]^* (\text{out}(w)) \]

After replacing \( w \) with \( \gamma(l, z, \eta^\Delta, f) \) and applying unfolding we obtain

\[ \psi^+(l, \gamma(l, z, \eta^\Delta, f)) = [\eta^\Delta, \lambda(z, u). (\text{out} \gamma)^t(l, z, u)]^* (\text{out} \gamma)(l, z, \eta^\Delta, f) = (\text{out} \gamma)^t(l, z, \eta^\Delta, f). \]

which allows us to reduce the goal (35) down to the following:

\[ [\eta^\Delta, \lambda x. \gamma^\Delta_2(l, x)] = \lambda z. (\text{out} \gamma)^t(l, z, \eta^\Delta, f). \]

It is easy to check that

\[ (\text{out} \gamma)(l, z, \eta^\Delta, f) = T_\Delta (\text{id} + \lambda z. (z, \eta^\Delta, f))[\eta^\Delta \text{inl}, \lambda x. (T_\Delta \text{inr}) \gamma_2(l, x)](z), \]

and therefore, by uniformity

\[ (\text{out} \gamma)^t(l, z, \eta^\Delta, f) = [\eta^\Delta \text{inl}, \lambda x. (T_\Delta \text{inr}) \gamma_2(l, x)]^t(z), \]

which allows us to replace (36) with

\[ [\eta^\Delta, \lambda x. \gamma^\Delta_2(l, x)] = [\eta^\Delta \text{inl}, \lambda x. (T_\Delta \text{inr}) \gamma_2(l, x)]^t. \]

The latter now follows by dinaturality:

\[ [\eta^\Delta \text{inl}, \lambda x. (T_\Delta \text{inr}) \gamma_2(l, x)]^t = ([(\eta^\Delta \text{inl}, \lambda x. (T_\Delta \text{inr}) \gamma_2(l, x))^* \eta^\Delta]^t = \]

\[ = [\eta^\Delta, ([(\eta^\Delta \text{inl}, \eta^\Delta)^* \lambda x. (T_\Delta \text{inr}) \gamma_2(l, x)]^t)^* \eta^\Delta = \]

\[ = [\eta^\Delta, (\lambda x. \gamma_2(l, x)]^t] = \]

\[ = [\eta^\Delta, \lambda x. \gamma^\Delta_2(l, x)]. \]
Proof of Proposition 19

Let us introduce the following notation:

\[ p(h/f) = \text{defrec } f(x)@v = \text{do } y \leftarrow h(x); v(y) \text{ in } p. \]

Note that the return type of \( h \) must be \( A \to |B|_{\Delta} \) with \( \Delta \) such that \( f \in \Delta \), to make the above expressions type. We shall occasionally abuse the notation and write \( p(h/f) \) also with \( h : A \to |B|_{\Delta} \) and \( f \notin \Delta \) assuming an obvious type casting from \( A \to |B|_{\Delta} \) to \( A \to |B|_{\Delta,.f} \).

Lemma 26. Suppose,

\[ \gamma_1 = [\Delta, f : A \to B; \Gamma \vdash p : C] : X \to T_a^b Z \]
\[ \gamma_2 = [\Gamma \vdash h : A \to |B|_{\Delta,.f}] : X \to (a \to T_a^b b) \]

where \( a = A, b = B, X = \Gamma \) and \( Z = C \). Then \( [p(h/f)] = \xi^t(\text{id}, \gamma_1) \) and \( \xi \) is given as follows:

\[ X \times T_a^b Z \xrightarrow{(\text{Tdist})(\text{id} \times \text{out})} T(X \times Z + X \times a \times (T_a^b Z)^b) \]
\[ \xrightarrow{\text{T}(\text{pr}_2 + \lambda(x,y,g), g^v(\gamma_2(x)(y)))} T(Z + T_a^b Z). \]

**Proof:** This becomes obvious as soon as we expand out all the definitions: We have

\[ p(h/f) = (\text{defrec } f(x)@v = \text{do } y \leftarrow \text{force } h(x); v(y) \text{ in } p). \]

Let \( \gamma_3 : X \times a \times T_a^b Z^b \to T_a^b Z^a \) be the semantics of the defining term for \( f \), then

\[ \gamma_3(x, y, g) = g^v(\gamma_2(x)(y)) \]

The claim follows by the definition of the semantics of recursive handling.

We rely on the following properties of recursive handlers.

Lemma 27. For any appropriately typed \( p, q \) and \( h \) and any \( f, g : A \to B \) from their effect contexts:

\[ p\{f/g\}\{\lambda x. \text{thunk}(h(x)\{f/g\})/f\} = p\{h/g\}\{\lambda x. \text{thunk}(h(x)\{h/g\})/f\}. \]

**Proof:** We assume for simplicity that the variable context \( \Gamma \) is empty. By Lemma 26 we have to show that \( \xi_2^t \zeta^t = \xi_3^t \xi_1^t \)

where

\[ \xi = [\eta^v \text{ inl, out}^1 \eta \text{ inr}(\eta^\nu \text{ inr})]^b \text{ out} : (T_a^b Z \to T_a^b (Z + (T_a^b Z)^b)) \]
\[ \xi_1 = T_a^b (\text{id} + \lambda(y, g). (g^v \gamma_1)(y)) \text{ out} : (T_a^b Z \to T_a^b (Z + (T_a^b Z)^b)) \]
\[ \xi_2 = T(\text{id} + \lambda(y, g). (g^v \gamma_2)(y)) \text{ out} : T_a^b Z \to T(Z + T_a^b Z) \]
\[ \xi_3 = T(\text{id} + \lambda(y, g). (g^v \gamma_3)(y)) \text{ out} : T_a^b Z \to T(Z + T_a^b Z) \]

and \( \gamma_1 = [h] : a \to (T_a^b)^b b, \gamma_2 = [\lambda x. h(x)\{f/g\}] : a \to T_a^b b, \gamma_3 = [\lambda x. h(x)\{h/g\}] : a \to T_a^b b. \)

Let \( u = \text{out}^1 \eta \text{ inr}(\text{id, \lambda, \eta^\nu}) : a \to T_a^b b \) and observe the following:

\[ [\eta^v \text{ inl, } \lambda(x, g). T_a^b (\text{inr } g) u(x)]^b \text{ out} \]
\[ = [\eta^v \text{ inl, } \lambda(x, g). T_a^b (\text{inr } g) \text{ out}^1 \eta \text{ inr}(x, \eta^\nu)]^b \text{ out} \]
\[ = [\eta^v \text{ inl, } \lambda(x, g). \text{ out}^1 T(\text{inr } g + (T_a^b (\text{inr } g))^b)] \eta \text{ inr}(x, \eta^\nu)]^b \text{ out} \]
\[ = [\eta^v \text{ inl, } \lambda(x, g). \text{ out}^1 \eta \text{ inr}(x, T_a^b (\text{inr } g)^b) \eta^v] \text{ out} \]
\[ = [\eta^v \text{ inl, } \lambda(x, g). \text{ out}^1 \eta \text{ inr}(x, \eta^\nu \text{ inr } g)]^b \text{ out} \]
\[ = [\eta^v \text{ inl, } \text{ out}^1 \eta \text{ inr}(\text{id, } \eta^\nu \text{ inr})]^b \text{ out} \]
\[ = [\eta^v \text{ inl, } \text{ out}^1 \eta \text{ inr}(\eta^\nu \text{ inr})]^b \text{ out} \]
\[ = \zeta \]

which means that according to Lemma 14 \( \zeta^t \) is a complete Elgot monad morphism induced by \( u \), in particular, \( \zeta \) is natural in \( Z \). Using this fact and slightly abusing the notation, note that \( \gamma_2 = \zeta^t \gamma_1 \). Analogously, by Lemma 25, \( \xi_1^t, \xi_2^t \) and \( \xi_3^t \) are complete Elgot monad morphism, in particular, as above, we have \( \gamma_3 = \xi_1^t \gamma_1 \).

Our further strategy is to split the target equation \( \xi_2^t \xi_1^t = \xi_3^t \xi_1^t \) into two:

\[ \xi_2^t \xi_1^t = w^t \quad \text{and} \quad w^t = \xi_3^t \xi_1^t \quad (38) \]
where $w$ is as follows

\[
\begin{align*}
(T^a_b Z)_a \xrightarrow{\text{out}} T^a_b (Z + ((T^a_b Z)_a) \\
\xrightarrow{T(id + (\text{out}^a_b)^\dagger)} T(Z + ((T^a_b Z)_a) + ((T^a_b Z)_a) \\
\xrightarrow{T(id + \lambda(y, g). (y^b \gamma_1)(y)) T[id, \text{inr}]^{-1}} T(Z + (T^a_b Z).)
\end{align*}
\]

Let us show commutativity of the following diagram

\[
\begin{array}{ccc}
(T^a_b Z) & \xrightarrow{\zeta^\dagger} & T^a_b Z \\
\downarrow w & & \downarrow \xi_2 \\
T(Z + (T^a_b Z) & \xrightarrow{T(id + \zeta^\dagger)} & T(Z + T^a_b Z)
\end{array}
\]

(39)

We calculate

\[
\begin{align*}
T(id + \zeta^\dagger) w &= T(id + \zeta^\dagger) T(id + \lambda(y, g). (g^b \gamma_2)(y)) T[id, \text{inr}] T(id + (\text{out}^a_b)^\dagger) \text{ out out} \\
&= T(id + \lambda(y, g). (\zeta^\dagger g^b \gamma_2)(y)) T[id, \text{inr} (\text{out}^a_b)^\dagger] \text{ out out} \\
&= T(id + \lambda(y, g). (\zeta^\dagger g^b \gamma_2)(y)) T[id, \text{inr} (\text{out}^a_b)^\dagger] \text{ out out} \\
&= T(id + \lambda(y, g). (g^b \gamma_2)(y)) T(id + (\zeta^\dagger)^\dagger) T[id, \text{inr} (\text{out}^a_b)^\dagger] \text{ out out} \\
&= \xi_2 \text{ out out}^{-1} T[id + (\zeta^\dagger)^\dagger] T[id, \text{inr} (\text{out}^a_b)^\dagger] \text{ out out} \\
&= \xi_2 \text{ out out}^{-1} T[id + (\zeta^\dagger)^\dagger, \text{inr}(\zeta^\dagger \text{ out}^a_b)^\dagger] \text{ out out}.
\end{align*}
\]

In order to finish the proof of (39), we are left to show that $\text{out}^{-1} T[id + (\zeta^\dagger)^\dagger, \text{inr}(\zeta^\dagger \text{ out}^a_b)^\dagger] \text{ out out} = \zeta^\dagger$. To this end, we first establish the following auxiliary identity:

\[
\begin{align*}
\zeta^\dagger \text{ out}^{-1} &= [\eta^\nu, \zeta^\dagger]^b_\alpha [\eta^\nu \text{ inl, out}^{-1} \eta \text{ inr} (\eta^\nu \text{ inr})_a^b]^\dagger \\
&= [\eta^\nu, [\eta^\nu, \zeta^\dagger]^b_\alpha \text{ out}^{-1} \eta \text{ inr} (\eta^\nu \text{ inr})_a^b]^\dagger \\
&= [\eta^\nu, \text{ out}^{-1} \eta \text{ inr} (\eta^\nu, \zeta^\dagger]^b_\alpha) (\eta^\nu \text{ inr})_a^b]^\dagger \\
&= [\eta^\nu, \text{ out}^{-1} \eta \text{ inr} (\zeta^\dagger)^b_\alpha]^\dagger.
\end{align*}
\]

(40)

Then we have

\[
\begin{align*}
\zeta^\dagger &= [\eta^\nu, \text{ out}^{-1} \eta \text{ inr} (\zeta^\dagger)^b_\alpha]^\dagger \text{ out} \\
&= \text{ out}^{-1} \text{ out}^{-1} \eta \text{ inr} (\zeta^\dagger)^b_\alpha, \eta \text{ inr} (\eta^\nu, \text{ out}^{-1} \eta \text{ inr} (\zeta^\dagger)^b_\alpha)]^\dagger \text{ out out} \\
&= \text{ out}^{-1} \text{ out}^{-1} \eta \text{ inr} (\zeta^\dagger)^b_\alpha, \eta \text{ inr} (\zeta^\dagger \text{ out}^a_b)^\dagger \text{ out out} 
\end{align*}
\]

(by 40)

After applying uniformity to (39), we obtain the right-hand side identity of (38).

Let us stick to the right-hand side of (38). Let $u_1 = \lambda(y, g). (g^b \gamma_1)(y)$ and $t = T[(\text{inl} + u_1), \text{inr} (\text{out}^a_b)^\dagger] \text{ out out}$. We rewrite $w^\dagger$ as follows:

\[
\begin{align*}
w^\dagger &= (T(id + u_1) T[id, \text{inr}] T(id + (\text{out}^a_b)^\dagger) \text{ out out})^\dagger \\
&= (T[id, \text{inr}] T[(\text{inl} + u_1), \text{inr} u_1 (\text{out}^a_b)^\dagger] \text{ out out})^\dagger \\
&= ((T[(\text{inl} + u_1), \text{inl} u_1 (\text{out}^a_b)^\dagger] \text{ out out})^\dagger)^\dagger \\
&= (T[id + u_1] (T[(\text{inl} + u_1), \text{inr} (\text{out}^a_b)^\dagger] \text{ out out})^\dagger)^\dagger \\
&= (T(id + u_1) t^\dagger)^\dagger \\
&= (\eta \text{ inl, } \eta \text{ inr} u_1)^* t^\dagger \\
&= [\eta, (\eta \text{ inl, } t^\dagger)^* \eta \text{ inr} u_1]^* t^\dagger \\
&= [\eta, (t^\dagger u_1)^* t]^\dagger \\
&= (T(id + t^\dagger u_1)) t^\dagger. 
\end{align*}
\]

(Lemma 28)
The proof of the latter is captured by the corresponding lemma whose proof is postponed. We compose the proof of the identity \( w^\dagger = \xi^1_3 \xi^1_1 \) as follows: \( w^\dagger = (T(id+t^1 u_1))^\dagger t^\dagger = \xi^1_3 \text{out}^\dagger T(id + (\xi^1_1)^{\dagger}_a) t^\dagger = \xi^1_3 \xi^1_1 \) where we in the obvious way used the following auxiliary identities:

\[
\text{out} \xi^1_1 = T(id + (\xi^1_1)^{\dagger}_a) t^\dagger \\
(T(id + t^1 u_1))^\dagger = \xi^1_3 \text{out}^\dagger T(id + (\xi^1_1)^{\dagger}_a)
\]

Let us show (41). Let \( \pi = T[\text{inl}, \text{inr}, \text{inl}, \text{inr}] \) and observe that

\[
\begin{align*}
\pi \text{out} \xi^1_1 &= \pi \text{out} T^b_a(id + u_1) \\
 &= \pi T((id + u_1) + T^b_a(id + u_1)^{\dagger}) \text{out} \\
 &= T([\text{inl}, \text{inr}], \text{inl} \text{inr} T^b_a(id + u_1)^{\dagger}) \text{out} \\
 &= T((id + (\xi^1_1)^{\dagger}_a) + id) t.
\end{align*}
\]

Now, by the definition of iteration, we immediately have

\[
\begin{align*}
\text{out} \xi^1_1 &= \text{out}[\eta, \xi^1_1] \text{out} \xi^1_1 \\
 &= [\text{out}[\eta, \xi^1_1], \eta \text{inr}([\eta, \xi^1_1]^{\dagger}) T(id + id)(\pi \text{out} \xi^1_1) t^\dagger \\
 &= T(id + ((\eta, \xi^1_1)^{\dagger} a) \text{id} (id + (\xi^1_1)^{\dagger}_a) t^\dagger \\
 &= T(id + (\xi^1_1)^{\dagger}_a) t^\dagger.
\end{align*}
\]

Finally, let us show (42). Let \( u_3 = \lambda(y, g). (g^b \gamma_3) (y) \) and consider the following diagram

\[
\begin{align*}
T(Z + ((T^b_a)^{b} Z_a)^{b}) &\xrightarrow{T(id + ((\xi^1_1)^{\dagger}_a) T(id + (\xi^1_1)^{\dagger}_a))} T(Z + ((T^b_a)^{b} Z_a)^{b}) &\xrightarrow{\text{out}^\dagger} &T^b_a Z \\
\downarrow T(id + u_1) & & &\downarrow T(id + u_3) \\
T(Z + (T^b_a)^{b} Z_a) &\xrightarrow{T(id + (\xi^1_1)^{\dagger})} T(Z + T^b_a Z_a) &\xrightarrow{\xi^1_3} &
\end{align*}
\]

which implies (42) by uniformity. The right triangle in (43) commutes by definition, the bottom triangle commutes by (41). Let us show commutativity of the inner quadrangle. It is equivalent to the following equation

\[(\xi^1_1 g)^* \gamma_3 = \xi^1_1 g^* \gamma_1\]

for any \( g : b \rightarrow T^b_a Z_a \). Recall that \( \gamma_3 = \xi^1_3 \gamma_1 \), hence we can simplify down to \( (\xi^1_1 g)^* \gamma_1 = \xi^1_1 g^* \). The latter property holds in particular if \( \xi^1_1 \) is a monad morphism, which it is by Lemma 25.

\( \blacksquare \)

**Lemma 28.** Let \( f : X \rightarrow T(Y + X) \). Then \( \eta, f^\dagger = (T(id + f))^\dagger \).

**Proof:** Consider the following trivially commuting diagram

\[
\begin{align*}
X &\xrightarrow{f} T(Y + X) \\
\downarrow f & & \downarrow T(id + f) \\
T(Y + X) &\xrightarrow{T(id + f)} T(Y + T(Y + X))
\end{align*}
\]

By uniformity, this implies \( f^\dagger = (T(id + f))^\dagger f \). Therefore \( \eta, f^\dagger = \eta, (T(id + f))^\dagger f^\dagger = \eta, (T(id + f))^\dagger T(id + f) = (T(id + f))^\dagger \) and we are done.

\( \blacksquare \)

**Lemma 29.** Let \( p \) satisfy the assumption that \( f \) does not occur in \( p \) and \( p \) may only contain \( w \) in the form \( \text{force}(w(t)) \) and not under thunks. Then

\[
p[f/w]\{w/f\} = p,
\]

**Proof:** By Lemma 26,

\[
\Delta; \Gamma, w : A \rightarrow [B] \Delta \vdash_c p[f/w]\{w/f\} : C] = \xi^1_3 (id, \gamma_1)
\]

40
where
\[
\xi(z, h, s) = T_\Delta (id + \lambda(y, g). g^b (\gamma_2(z, h)(y))) \text{ out}(s),
\]
\[
\gamma_1 = \llbracket \Delta ; \Gamma, w : A \rightarrow [B]_\Delta \vdash_c p[f/w] : C \rrbracket,
\]
\[
\gamma_2 = \llbracket \Delta ; \Gamma, w : A \rightarrow [B]_\Delta \vdash_v w : A \rightarrow [B]_\Delta \rrbracket.
\]
After obvious simplifications we obtain
\[
\xi(z, h, s) = T_{\Delta'} (id + \lambda(y, g). g^b h(y)) \text{ out}(s).
\]
where \( \Delta = [\Delta', f : A \rightarrow B] \). According to Lemma 25, the resulting morphism \( \lambda s. \xi^t(z, h, s) \) is a complete Elgot monad morphism, which means that the operation \( \{w/f\} \) distributes over the first-order structures of the language, and hence (44) follows by structural induction over \( p \). Let us consider the most representative cases.

- \( p \) does not depend on \( w \) (Induction Base). Then \([p]\) factors through \( \text{pr}_1 : \Gamma \times (A \rightarrow T_\Delta B) \rightarrow \Gamma \) and \( p[f/w] = p \). The latter means that \( \gamma_1 = \text{ext}[p] \). Therefore \([p[f/w] \{w/f\}] = \xi^t (\text{id}, \text{ext}[p]) = \xi^t [p] \).

- \( p = \text{thunk}(w(t)) \) for some \( t \) (Induction Base). Note that for typing reasons \( w \) can not occur in \( t \), hence we use \([t]\) as a shortening for \([\Gamma \vdash_v t : A] \). Note that, by definition, \( \text{thunk}(w(t)) = \text{out}^t \text{inr} \langle [t] \rangle \text{pr}_1, \lambda, \eta^{\Delta'} \rangle \). It is then easy to verify that \([p[f/w] \{w/f\}] = \xi^t (\text{id}, \text{out}^t \text{inr} \langle [t] \rangle \text{pr}_1, \lambda, \eta^{\Delta'} \rangle) = \lambda(z, h). h(\text{ext}[f](z)) = \text{thunk}(w(t)) = [p] \).

- \( p = (\text{do } s \leftarrow s; t) \) (Induction Step). If \( x \neq w \) then, by induction hypothesis, \( p[f/w] \{w/f\} = \text{do } s \leftarrow s; t = p \). If \( x = w \) then for typing reasons \( s \) can not contain \( w \); then w.l.o.g. \( p \) does not contain \( w \) and we are done by the induction base.

- \( p = (\text{def } g(x)@v = v' \text{ in } q' \text{ in } q) \) (Induction Step). Analogously to the previous case, we will be done once we show that
\[
(\text{def } g(x)@v = v' \text{ in } q') \{w/f\} = (\text{def } g(x)@v = v' \text{ in } q \text{ in } q' \{w/f\})
\]
where \( q' = q[f/w], r' = r[f/w] \).
\[
\gamma_1 = \llbracket \Delta, g : D \rightarrow E ; \Gamma, w : A \rightarrow [B]_{\Delta, g} \vdash_c q' : C \rrbracket,
\]
\[
\gamma_2 = \llbracket \Delta, g : D \rightarrow E ; \Gamma, w : A \rightarrow [B]_{\Delta, g}, x : D, v : D \rightarrow [C]_{\Delta, g} \vdash_c r' : C \rrbracket.
\]
where we assume that \( \Delta = [\Delta', f : A \rightarrow B] \). Let moreover
\[
\gamma'_1 = \llbracket \Delta, g : D \rightarrow E ; \Gamma \vdash_c q' : C \rrbracket = \lambda z. \gamma_1(z, \lambda x. \sigma \text{ out}^{\eta^{\Delta'.g}} \text{ inr}(x, \lambda, \eta^{\Delta'.g})) \]
\[
\gamma'_2 = \llbracket \Delta, g : D \rightarrow E ; \Gamma, x : D, v : D \rightarrow [C]_{\Delta, g} \vdash_c r' : C \rrbracket = \lambda (z, x, h). \gamma_2(z, \lambda x. \sigma \text{ out}^{\eta^{\Delta'.g}} \text{ inr}(x, \lambda, \eta^{\Delta'.g})) (x, h).
\]
where \( \sigma \) is the obvious natural isomorphism between \( T_{\Delta, f, g} \) and \( T_{\Delta, g, f} \). Then
\[
([\text{def } g(x)@u = r' \text{ in } q' \{w/f\}])(z) = \xi^{t}[z, \sigma(\eta^{\Delta'.g}), \text{id}^b \text{ ext } \psi(z, \gamma'_1(z))]
\]
where \( \xi \) is as above and
\[
\psi(z, s) = T_\Delta (id + \lambda(x, h). \gamma'_2(z, x, h)) \text{ out}(s).
\]
Using the fact that \( \lambda s. \xi^t(z, h, s) \) is a complete Elgot monad morphism, we calculate
\[
([\text{def } g(x)@v = v' \text{ in } q' \{w/f\}])(z)
\]
\[
= [\eta^{\Delta'.g}, \lambda s. \xi^t(z, s)]^b \xi^t(z, \sigma \text{ ext } \psi(z, \gamma'_1(z))]
\]
\[
= [\eta^{\Delta'.g}, \text{id}^b T_{\Delta', g}(id + \lambda s. \xi^t(z, s))]^t \xi^t(z, \sigma \text{ ext } \psi(z, \gamma'_1(z))]
\]
\[
= [\eta^{\Delta'.g}, \text{id}^b T_{\Delta', g}(id + \lambda s. \xi^t(z, s))]^b \text{ ext } \xi^t(z, \psi(z, \gamma'_1(z))]
\]
\[
= [\eta^{\Delta'.g}, \text{id}^b \text{ ext } T_{\Delta}(id + \lambda s. \xi^t(z, s))]^t \xi^t(z, \psi(z, \gamma'_1(z))]
\]
\[
= [\eta^{\Delta'.g}, \text{id}^b \text{ ext } T_{\Delta}(id + \lambda(x, h), \xi^t(z, \gamma'_2(z, x, h))) \text{ out } \gamma'_1(z)]
\]
\[
= [\eta^{\Delta'.g}, \text{id}^b \text{ ext } T_{\Delta}(id + \lambda(x, h), \xi^t(z, \gamma'_2(z, x, h))) \text{ out } \gamma'_1(z)]
\]
\[
T_{\Delta'}(id + \lambda h. \lambda x. \xi^t(z, \sigma(h(x)))) \xi^t(z, \text{ out } \gamma'_1(z))
\]
\[ \eta \Delta', g, \text{id} \text{ ext } T_{\Delta'}(\text{id} + \lambda(x, h), \xi^1(z, \gamma'_2(z, x, h \sigma \text{ ext}))) \text{ out } \xi^1(z, \sigma \gamma'_1(z)) \]

\[ \text{def } g(x)@v = r'\{w/f\}/v\{w/f\} \text{ in } q'\{w/f\}(z) \]

- \( p = (\text{defrec } g(x)@v = h \text{ in } q) \) (Induction Step). Like in the previous case, this amounts the the analogous verification of

\[ (\text{defrec } g(x)@u = r' \text{ in } q')\{w/f\} = (\text{defrec } g(x)@v = r \text{ in } q). \]

where \( q' = q[f/w], r' = r[f/w]. \)

The proof of Proposition 19 now runs as follows:

\[
\begin{align*}
\text{letrec } &w(x) = q \text{ in } p) \\
&= p[f/w]\{\lambda x. q[f/w]/f\} \quad \text{(definition)} \\
&= p[g/w]\{f/g\}\{\lambda x. q[f/w]/f\} \quad \text{(Lemma 29)} \\
&= p[g/w]\{\lambda x. q[f/w]/g\}\{\lambda x. q[f/w]/f\} \quad \text{(Lemma 27)} \\
&= p[g/w]\{\lambda x. q/g\}[f/w]\{\lambda x. q[f/w]/f\} \quad \text{(Lemma 29)} \\
&= p[\lambda x. q/g][w/f]\{\lambda x. q[f/w]/f\} \quad \text{(Lemma 29)} \\
&= (\text{letrec } w = q \text{ in } p[\lambda x. q/w]) \quad \text{(definition)}
\end{align*}
\]

where \( g \) is another free effect symbol not occurring in the effect contexts of \( p \) and \( q \) and distinct from \( f \).

**Proof of Proposition 20**

We prove this equation on the syntactic level:

\[
\text{letrec } w(z) = \text{ case } z \text{ of } \text{inl } y \rightarrow \text{ ret } y; \text{ inr } x \rightarrow (\text{ do } z \leftarrow q; w(z)) \text{ in }
\]

\[
\text{do } z \leftarrow p; w(z)
\]

\[
= \text{ defrec } f(x)@v = \text{ do }
\]

\[
y \leftarrow (\text{ case } z \text{ of } \text{inl } y \rightarrow \text{ ret } y; \text{ inr } x \rightarrow (\text{ do } z \leftarrow q; w(z))(x))[f/w];
\]

\[
v(y) \text{ in } \text{ do } z \leftarrow p[f/w]; f(z).
\]

Since \( p \) does not mention \( w \), this it equal to

\[
\text{defrec } f(x)@v = \text{ do } y \leftarrow (\text{ case } x \text{ of } \text{inl } y \rightarrow v(y); \text{ inr } x \rightarrow (\text{ do } z \leftarrow q; y \leftarrow f(z)); v(y)) \text{ in } \text{ do } z \leftarrow p; f(z)
\]

\[
= \text{ defrec } f(x)@v = \text{ case } x \text{ of } \text{inl } y \rightarrow v(y); \text{ inr } x \rightarrow (\text{ do } z \leftarrow q; y \leftarrow f(z); v(y)) \text{ in } \text{ do } z \leftarrow p; f(z).
\]

We are done by Equation (*).