Memoryful Geometry of Interaction II

Recursion and Adequacy

Koko Muroya
University of Tokyo, Japan
muroykk@is.s.u-tokyo.ac.jp

Naohiko Hoshino
RIMS, Kyoto University, Japan
naophiko@kurims.kyoto-u.ac.jp

Ichiro Hasuo
University of Tokyo, Japan
ichiro@is.s.u-tokyo.ac.jp

Abstract

A general framework of *Memoryful Geometry of Interaction (mGoI)* is introduced recently by the authors. It provides a sound translation of lambda-terms (on the high-level) to their realizations by stream transducers (on the low-level), where the internal states of the latter (called memories) are exploited for accommodating *algebraic effects* of Plotkin and Power. The translation is compositional, hence "denotational," where transducers are inductively composed using an adaptation of Barbosa's coalgebraic component calculus.

In the current paper we extend the mGoI framework and provide a systematic treatment of *recursion*—an essential feature of programming languages that was however missing in our previous work. Specifically, we introduce two new fixed-point operators in the coalgebraic component calculus. The two follow the previous work on recursion in GoI and are called *Girard style* and *Mackie style*: the former obviously exhibits some nice domain-theoretic properties, while the latter allows simpler construction. Their equivalence is established on the categorical (or, traced monoidal) level of abstraction, and is therefore generic with respect to the choice of algebraic effects. Our main result is an adequacy theorem of our mGoI translation, against Plotkin and Power's operational semantics for algebraic effects.

Categories and Subject Descriptors D.3 [*Formal Definitions and Theory*]: Semantics; F.3 [*Semantics of Programming Languages*]: Algebraic approaches to semantics

General Terms Theory

Keywords Geometry of Interaction, monad, algebraic effect, adequacy

1. Introduction

Geometry of interaction (GoI) introduced by Girard is a semantics of linear logic [11]. The aim of GoI project is to mathematically understand dynamical process of cut-elimination. In other words, via Curry-Howard correspondence, GoI project aims to mathematically understand dynamical process of program execution. This is like game semantics that interprets programs by dialogue between two players [1, 17].

This paper is a second step towards our prospect of developing GoI-based compilation technique for effectful programming languages. Our prospect is motivated by Ghica's Geometry of Synthesis [7] where Ghica applied the idea of GoI and game semantics to give a semantics-directed compiler. In our first step [16], we introduced GoI semantics called *memoryful GoI* (mGoI) for Moggi's computational lambda calculus equipped with algebraic effects. Our goal in this paper is to show that mGoI introduced in our previous work can accommodate recursion. Main contributions of this paper are as follows.

- We give two mGoI interpretation of recursion.
  - One is Girard style interpretation. Interpretation of recursion is given as limits of approximating chains as given in [12]. This modeling style is much like the one in domain theory and is easy to mathematically deal with.
  - The other is Mackie style interpretation. Interpretation of recursion is given by feedback loops as given in [22]. Mackie style is much simpler than Girard style and is easy to implement.
- We show that these two interpretations of recursion are the same. Coincidence of these styles enables us to enjoy advantages of each of them. In this paper, we give interpretation of recursion in Mackie style because of its simplicity, and we use Girard style to show adequacy of mGoI interpretation.
- We prove adequacy of mGoI interpretation with respect to the operational semantics given by Plotkin and Power [25].

Our framework is applicable to a wide class of computational effects. For example, mGoI can model nondeterministic choice, probabilistic choice, global states, interactive I/O, exceptions and their combinations. The generality of mGoI is achieved by a categorical framework of GoI called *GoI situation* introduced by Abramsky, Haghverdi and Scott in [2].

Construction of mGoI semantics for the computational lambda calculus is as follows:

$$\begin{array}{c}\text{A monad } T\colon \mathbf{Set} \to \mathbf{Set}\\ \downarrow \\ \text{The Kleisli category } \mathbf{Set}_T \\ \downarrow \\ T\text{-transducers} \\ \downarrow \text{GoI situation} \\ \text{mGoI semantics for the computational } \lambda\text{-calculus}\end{array}$$

Our starting point is a monad on the category $\mathbf{Set}$ of sets and functions. Given a monad $T\colon \mathbf{Set} \to \mathbf{Set}$, we construct a category of sets and T-transducers. Roughly speaking, T-transducers are "T-branching" Mealy machines: when T is the powerset monad $\mathcal{P}$, a $\mathcal{P}$-transducer is a nondeterministic Mealy machine; when T is the subdistribution monad $\mathcal{D}$, a $\mathcal{D}$-transducer is a probabilistic Mealy machine; when T is an exception monad $\mathcal{E}$, a $\mathcal{E}$-transducer is a Mealy machine that may raise errors.

By constructing a GoI situation on the category of sets and T-transducers, we obtain GoI semantics for the lambda calculus. What we clarified in our previous work is that the monad T induces interpretation of computational effects. For example, we interpret nondeterministic choice $\mathtt{0} \sqcup \mathtt{1}$ by a $\mathcal{P}$-transducer, which is a nondeterministic Mealy machine, whose transition diagram is given by

$$\begin{tikzcd} & {} \arrow[d] & \\ s_l \arrow[loop left, "q/0"] & s_0 \arrow[l, "q/0"'] \arrow[r, "q/1"] & s_r \arrow[loop right, "q/1"] \end{tikzcd} .$$

We explain how this $\mathcal{P}$-transducer behaves by using mGoI interpretation of $(\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x})\,(\mathtt{0} \sqcup \mathtt{1})$. Interpretation of this term is given by the following interactive communication between $\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x}$ and $\mathtt{0} \sqcup \mathtt{1}$:

1. $\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x}$ requires output of the left argument $\mathtt{x}$.
2. $\mathtt{0} \sqcup \mathtt{1}$ nondeterministically answers $\mathtt{0}$ or $\mathtt{1}$ and stores its choice.
3. $\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x}$ requires output of the right argument $\mathtt{x}$.
4. $\mathtt{0} \sqcup \mathtt{1}$ answers its first choice.
5. $\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x}$ outputs the sum of the first answer and the second answer from $\mathtt{0} \sqcup \mathtt{1}$.

As a result, $\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x}$ nondeterministically outputs $\mathtt{0}$ or $\mathtt{2}$. This is exactly the call-by-value evaluation result of $(\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x})\,(\mathtt{0} \sqcup \mathtt{1})$. The important point in this dialogue is that $\mathtt{0} \sqcup \mathtt{1}$ memorizes its first choice by using states. Use of this memorization mechanism is essential. Without memorization, $(\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x})\,(\mathtt{0} \sqcup \mathtt{1})$ may output $\mathtt{1}$ because $\mathtt{0} \sqcup \mathtt{1}$ may answer $\mathtt{0}$ to the first question and $\mathtt{1}$ to the second question from $\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x}$. We note that we have similar problems for call-by-name situation.

1.1 Related Works

Recursion in GoI There already exist some works on interpreting recursion in GoI. The earliest one is Girard's GoI II [12]. Girard's idea is similar to the one in domain theory, i.e., fixed points are given as limits of approximating ω-chains. In this paper, we call this interpretation of recursion Girard style. Results in GoI II are generalized in terms of unique decomposition category in [13]. We can find another style of interpreting recursion in [22] where Mackie employed feedback loops to interpret recursion. We call this interpretation of recursion Mackie style. We can also find Mackie style fixed point operator in [15, 19]. In this paper, we show that these two styles are the same in mGoI semantics.

GoI as Mathematical Framework for Implementation While the original motivation of GoI stems from mathematical investigation in proof theory, the dynamical aspect of GoI provides practical applications. In GoI semantics, we model programs by means of interactive dialogue, and we can understand GoI semantics as translation of high-level languages into low-level languages. Danos and Regnier presented this basic idea of applying GoI to implementation of functional programming languages in [4]. Developing their idea, Mackie gave an implementation of PCF [22] where implementation of PCF is given by simulating interactive dialogue by means of an assembly language. GoI-based implementation inherits some features of GoI: simple run-time systems and compositional interpretation of programs. Furthermore, soundness of GoI semantics assures correctness of GoI-based implementation by construction. Recently, in the series of works [7–10], Ghica applied the idea of GoI and game semantics to give a semantics-directed compiler for programming languages based on Reynolds' Syntactic Control of Interference. Ghica also pointed simplicity of compilation results of programs: circuits generated by compilation consist of a few basic circuits connected through wirings. Dal Lago and Schöpp applied GoI to design a functional programming language for computation with sublinear space [3]. Their design is based on an observation that space-efficient computation can naturally be organized into interactive dialogue.

Call-by-value and Computational Effects GoI semantics is call-by-name in nature: arguments are evaluated only if they are called in interactive dialogue. Therefore, in order to model call-by-value computation, we need to suitably adapt GoI interpretation. There are several approaches to call-by-value GoI. An early approach given in [6] is to employ jumping mechanism to enforce call-by-value evaluation. In [29], Schöpp employed cps-translation with some refinement to achieve efficient implementation of call-by-value computation. Later, this approach is reformulated in terms of typed closure conversion [30]. In this paper, we employ cps-translation, and therefore, our approach is close to Schöpp's approach (without refinement for efficiency). Recently, another approach to call-by-value GoI is proposed in [19].

Schöpp also studied combination of GoI and effects in [28], where he introduced GoI semantics for a functional programming language INTML[WEC]. We can embed a restriction of the enriched effect calculus, which is introduced by Egger, Møgelberg and Simpson [5], into INTML[WEC], which can accommodate computational effects through this embedding. Our basic idea in this paper is similar to Schöpp's approach: modeling computational effects by means of computational effects in interactive dialogue. Differences between our work and Schöpp's work are: while the computational lambda calculus allows to use effects at any type, it is not clear how to use computational effects at any type in INTML[WEC]; while control operators and locally scoped states can be modeled in Schöpp's approach, we restrict our attention to algebraic effects. Precise comparison is future work.

1.2 Organization of the Paper

First in Section 2 we describe our target language $\mathcal{L}_\Sigma$ and its operational semantics. They are slight adaptations of those introduced in [25]. In Section 3 we give algebraic operations on monads, that model algebraic effects in a uniform way. We also state the requirement of our framework on monads in this section. In Section 4 we recall the component calculus used in the mGoI framework. It is extended in Section 5 by two styles of fixed point operators. Their properties are investigated in this section as well. Finally we give a translation of the language $\mathcal{L}_\Sigma$ to transducers, as well as its adequacy, in Section 6; and illustrate the translation and execution of the resulting transducer in Section 7 using an example.

2. Target Language and Operational Semantics

This section is devoted to our target language $\mathcal{L}_\Sigma$ and its operational semantics. They are slight adaptations of those given by Plotkin & Power [25]; differences are introduction of coproduct types $\tau + \tau$ and replacement of the base type $\mathtt{bool}$ by a coproduct type $1 + 1$.

2.1 The Target Language $\mathcal{L}_\Sigma$

The language $\mathcal{L}_\Sigma$ extends the call-by-value computational λ-calculus [23] with algebraic operations, product/coproduct types and arithmetic primitives. It is parametrized by an algebraic signature Σ whose element is an *operation* $\mathtt{op}$ that comes with its *arity* $\mathrm{ar}(\mathtt{op})$. In this paper we assume all arities are finite.[1] For an operation $\mathtt{op} \in \Sigma$ we often write $\mathtt{op}^0$ if its arity is zero and $\mathtt{op}^+$ if positive.

Example 2.1. Here are examples of algebraic signatures taken from [26].

[1] Readers are referred to [26] for accommodation of countable arities.

- The signature $\Sigma_{\mathrm{except}} = \{\mathtt{raise}_e \mid e \in E\}$ is for *exceptions*: the set E specifies exceptions and each nullary operation $\mathtt{raise}_e()$ raises an exception e.
- The signature $\Sigma_{\mathrm{ndet}} = \{\sqcup\}$ is for *nondeterminism*, where the binary operation $\sqcup$ represents nondeterministic choice.
- The signature $\Sigma_{\mathrm{prob}} = \{\sqcup_p \mid p \in [0,1]\}$ is for *probabilistic choice*: in $\mathtt{M} \sqcup_p \mathtt{N}$, $\mathtt{M}$ is chosen with probability p; and $\mathtt{N}$ is with probability $1-p$.
- The signature $\Sigma_{\mathrm{glstate}} = \{\mathtt{lookup}_l \mid l \in \mathit{Loc}\} \cup \{\mathtt{update}_{l,v} \mid l \in \mathit{Loc}, v \in \mathit{Val}\}$ is for *actions on global states*, where Loc is a set of locations and Val is a finite set of values. In $\mathtt{lookup}_l(\mathtt{M}_{v_1}, \ldots, \mathtt{M}_{v_{|\mathit{Val}|}})$, one of the arguments is chosen according to a value stored by a global state of location l. In $\mathtt{update}_{l,v}(\mathtt{M})$, $\mathtt{M}$ is executed after updating a global state of location l with a value v.

The types τ, terms $\mathtt{M}$ and values $\mathtt{V}$ of $\mathcal{L}_\Sigma$ are defined by the following BNF's:

$$\begin{aligned}
\tau &::= \mathtt{unit} \mid \mathtt{nat} \mid \tau \to \tau \mid \tau \times \tau \mid \tau + \tau \\
\mathtt{M} &::= \mathtt{x} \in \mathbf{Var} \mid \lambda \mathtt{x} : \sigma.\, \mathtt{M} \mid \mathtt{M}\,\mathtt{M} \mid \mathtt{rec}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M} \\
&\quad \mid \mathtt{op}^{+}(\mathtt{M}_1, \ldots, \mathtt{M}_{\mathrm{ar}(\mathtt{op})}) \mid \mathtt{op}^{0}() \mid \underline{*} \mid \mathtt{fst}(\mathtt{M}) \mid \mathtt{snd}(\mathtt{M}) \mid \langle \mathtt{M}, \mathtt{M} \rangle \\
&\quad \mid \mathtt{inl}_{\tau,\sigma}(\mathtt{M}) \mid \mathtt{inr}_{\tau,\sigma}(\mathtt{M}) \mid \mathtt{case}(\mathtt{M}, \mathtt{x}.\, \mathtt{M}, \mathtt{y}.\, \mathtt{M}) \mid \underline{n} \in \mathbb{N} \mid \mathtt{M} + \mathtt{M} \\
\mathtt{V} &::= \mathtt{x} \in \mathbf{Var} \mid \lambda \mathtt{x} : \sigma.\, \mathtt{M} \mid \underline{*} \mid \langle \mathtt{V}, \mathtt{V} \rangle \mid \mathtt{inl}_{\tau,\sigma}(\mathtt{V}) \mid \mathtt{inr}_{\tau,\sigma}(\mathtt{V}) \\
&\quad \mid \underline{n} \in \mathbb{N}
\end{aligned}$$

where $\mathbf{Var}$ is a (countable) set of variables, $\mathbb{N}$ is the set of natural numbers and $\mathtt{op} \in \Sigma$ is an operation. Substitution $\mathtt{M}[\mathtt{N}/\mathtt{x}]$ is defined as usual and a term is called *closed* if it has no free variables.

A type judgment $\Gamma \vdash \mathtt{M} : \tau$, where $\Gamma = \mathtt{x}_1 : \tau_1, \ldots, \mathtt{x}_m : \tau_m$, is inductively defined by the typing rules shown in Figure 1. They are standard except those for operations in Σ; note that nullary operations $\mathtt{op}^0()$ can be arbitrarily typed. In the paper we assume any term $\mathtt{M}$ is well-typed, i.e. there exists a derivable type judgment $\Gamma \vdash \mathtt{M} : \tau$.

$$\frac{}{\Gamma \vdash \mathtt{x}_i : \tau_i} \quad \frac{\Gamma, \mathtt{x} : \sigma \vdash \mathtt{M} : \tau}{\Gamma \vdash \lambda \mathtt{x} : \sigma.\, \mathtt{M} : \sigma \to \tau} \quad \frac{\Gamma \vdash \mathtt{M} : \sigma \to \tau \quad \Gamma \vdash \mathtt{N} : \sigma}{\Gamma \vdash \mathtt{M}\ \mathtt{N} : \tau}$$

$$\frac{\Gamma, \mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma \vdash \mathtt{M} : \tau}{\Gamma \vdash \mathtt{rec}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M} : \sigma \to \tau}$$

$$\frac{\Gamma \vdash \mathtt{M}_i : \tau \ \ (i = 1, \ldots, \mathrm{ar}(\mathtt{op}))}{\Gamma \vdash \mathtt{op}^{+}(\mathtt{M}_1, \ldots, \mathtt{M}_{\mathrm{ar}(\mathtt{op})}) : \tau} \quad \frac{}{\Gamma \vdash \mathtt{op}^0() : \tau} \quad \frac{}{\Gamma \vdash \underline{*} : \mathtt{unit}}$$

$$\frac{\Gamma \vdash \mathtt{M} : \tau \times \sigma}{\Gamma \vdash \mathtt{fst}(\mathtt{M}) : \tau} \quad \frac{\Gamma \vdash \mathtt{M} : \tau \times \sigma}{\Gamma \vdash \mathtt{snd}(\mathtt{M}) : \sigma} \quad \frac{\Gamma \vdash \mathtt{M} : \tau \quad \Gamma \vdash \mathtt{N} : \sigma}{\Gamma \vdash \langle \mathtt{M}, \mathtt{N} \rangle : \tau \times \sigma}$$

$$\frac{\Gamma \vdash \mathtt{M} : \tau}{\Gamma \vdash \mathtt{inl}_{\tau,\sigma}(\mathtt{M}) : \tau + \sigma} \quad \frac{\Gamma \vdash \mathtt{M} : \sigma}{\Gamma \vdash \mathtt{inr}_{\tau,\sigma}(\mathtt{M}) : \tau + \sigma}$$

$$\frac{\Gamma \vdash \mathtt{M} : \tau + \tau' \quad \Gamma, \mathtt{x} : \tau \vdash \mathtt{N} : \sigma \quad \Gamma, \mathtt{x}' : \tau' \vdash \mathtt{N}' : \sigma}{\Gamma \vdash \mathtt{case}(\mathtt{M}, \mathtt{x}.\, \mathtt{N}, \mathtt{x}'.\, \mathtt{N}') : \sigma}$$

$$\frac{}{\Gamma \vdash \underline{n} : \mathtt{nat}} \quad \frac{\Gamma \vdash \mathtt{M} : \mathtt{nat} \quad \Gamma \vdash \mathtt{N} : \mathtt{nat}}{\Gamma \vdash \mathtt{M} + \mathtt{N} : \mathtt{nat}}$$

Figure 1: Typing Rules of $\mathcal{L}_\Sigma$

2.2 Operational Semantics

For $\mathcal{L}_\Sigma$ we define two kinds of operational semantics: the *small step* one and the *medium step* one. They are specified via decomposition of terms into evaluation contexts $\mathtt{E}$ and redexes $\mathtt{R}$, defined by the following BNF's.

$$\begin{aligned}
\mathtt{E} &::= [-] \mid \mathtt{E}\,\mathtt{M} \mid \mathtt{V}\,\mathtt{E} \mid \mathtt{fst}(\mathtt{E}) \mid \mathtt{snd}(\mathtt{E}) \mid \langle \mathtt{E}, \mathtt{M} \rangle \mid \langle \mathtt{V}, \mathtt{E} \rangle \\
&\quad \mid \mathtt{inl}_{\tau,\sigma}(\mathtt{E}) \mid \mathtt{inr}_{\tau,\sigma}(\mathtt{E}) \mid \mathtt{case}(\mathtt{E}, \mathtt{x}.\, \mathtt{M}, \mathtt{y}.\, \mathtt{M}) \mid \mathtt{E} + \mathtt{M} \mid \mathtt{V} + \mathtt{E}
\end{aligned}$$

$$\begin{aligned}
\mathtt{R} &::= (\lambda \mathtt{x} : \sigma.\, \mathtt{M})\ \mathtt{V} \mid \mathtt{rec}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M} \\
&\quad \mid \mathtt{op}^{+}(\mathtt{M}_1, \ldots, \mathtt{M}_{\mathrm{ar}(\mathtt{op})}) \mid \mathtt{fst}(\langle \mathtt{V}, \mathtt{V} \rangle) \mid \mathtt{snd}(\langle \mathtt{V}, \mathtt{V} \rangle) \\
&\quad \mid \mathtt{case}(\mathtt{inl}_{\tau,\sigma}(\mathtt{V}), \mathtt{x}.\, \mathtt{M}, \mathtt{y}.\, \mathtt{M}) \mid \mathtt{case}(\mathtt{inr}_{\tau,\sigma}(\mathtt{V}), \mathtt{x}.\, \mathtt{M}, \mathtt{y}.\, \mathtt{M}) \\
&\quad \mid \mathtt{V} + \mathtt{V}
\end{aligned}$$

If a closed term $\mathtt{M}$ is not a value, it is decomposed into either the form $\mathtt{E}[\mathtt{R}]$ or $\mathtt{E}[\mathtt{op}^0()]$, where $\mathtt{E}$, $\mathtt{R}$ and/or $\mathtt{op}^0$ are uniquely determined.

For closed redexes, transition relations are defined as in Figure 2. The unlabeled "pure" transition relation $\to$ corresponds to the ordinal β-reduction; and a labeled "effectful" transition relation $\overset{\mathtt{op}_i}{\to}$ represents reduction of a term $\mathtt{op}^{+}(\mathtt{M}_1, \ldots, \mathtt{M}_{\mathrm{ar}(\mathtt{op})})$ to one argument $\mathtt{M}_i$, according to an effect generated by $\mathtt{op}^{+}$. Additionally for a term $\mathtt{op}^0()$, a labeled "effectful termination" predicate $\downarrow_{\mathtt{op}}$ is defined by $\mathtt{op}^0() \downarrow_{\mathtt{op}}$.

$$\begin{aligned}
&(\lambda \mathtt{x} : \sigma.\, \mathtt{M})\ \mathtt{V} \to \mathtt{M}[\mathtt{V}/\mathtt{x}] \\
&\mathtt{rec}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M} \to (\lambda \mathtt{x} : \sigma.\, \mathtt{M})[\mathtt{rec}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M}/\mathtt{f}] \\
&\mathtt{fst}(\langle \mathtt{V}_1, \mathtt{V}_2 \rangle) \to \mathtt{V}_1 \qquad \mathtt{snd}(\langle \mathtt{V}_1, \mathtt{V}_2 \rangle) \to \mathtt{V}_2 \\
&\mathtt{case}(\mathtt{inl}_{\tau,\sigma}(\mathtt{V}), \mathtt{x}.\, \mathtt{N}, \mathtt{x}'.\, \mathtt{N}') \to \mathtt{N}[\mathtt{V}/\mathtt{x}] \\
&\mathtt{case}(\mathtt{inr}_{\tau,\sigma}(\mathtt{V}), \mathtt{x}.\, \mathtt{N}, \mathtt{x}'.\, \mathtt{N}') \to \mathtt{N}'[\mathtt{V}/\mathtt{y}'] \qquad \underline{n} + \underline{m} \to \underline{n + m} \\
&\mathtt{op}^{+}(\mathtt{M}_1, \ldots, \mathtt{M}_{\mathrm{ar}(\mathtt{op})}) \overset{\mathtt{op}_i}{\to} \mathtt{M}_i \ \ (i = 1, \ldots, \mathrm{ar}(\mathtt{op}))
\end{aligned}$$

Figure 2: Transition Relations for Closed Redexes

Using the transition relations and predicates, we define the small step operational semantics for a closed term, that is not a value, by

$$\frac{\mathtt{R} \to \mathtt{M}}{\mathtt{E}[\mathtt{R}] \to \mathtt{E}[\mathtt{M}]} \qquad \frac{\mathtt{R} \overset{\mathtt{op}_i}{\to} \mathtt{M}}{\mathtt{E}[\mathtt{R}] \overset{\mathtt{op}_i}{\to} \mathtt{E}[\mathtt{M}]} \qquad \frac{\mathtt{op}^0() \downarrow_{\mathtt{op}}}{\mathtt{E}[\mathtt{op}^0()] \downarrow_{\mathtt{op}}}$$

and define the medium step operational semantics for a closed term $\mathtt{M}$ by

$$\begin{aligned}
\mathtt{M} \Rightarrow \mathtt{V} &\overset{\mathrm{def.}}{\iff} \mathtt{M} \to^{*} \mathtt{V} \\
\mathtt{M} \overset{\mathtt{op}_i}{\Rightarrow} \mathtt{N} &\overset{\mathrm{def.}}{\iff} \exists \mathtt{L}.\, \mathtt{M} \to^{*} \mathtt{L} \overset{\mathtt{op}_i}{\to} \mathtt{N} \\
\mathtt{M} \Downarrow_{\mathtt{op}} &\overset{\mathrm{def.}}{\iff} \exists \mathtt{L}.\, \mathtt{M} \to^{*} \mathtt{L} \downarrow_{\mathtt{op}} \\
\mathtt{M} \Uparrow &\overset{\mathrm{def.}}{\iff} \exists \text{ infinite sequence } \mathtt{M} \to \mathtt{M}' \to \cdots .
\end{aligned}$$

Both small and medium step operational semantics are uniquely determined for each closed term.

Lemma 2.2. (I) Any closed term $\mathtt{M}$ of type τ satisfies just one of the following:

- $\mathtt{M} \equiv \mathtt{V}$ for a (unique closed) value $\mathtt{V}$;
- $\mathtt{M} \to \mathtt{N}$ for a unique closed term $\mathtt{N}$ of type τ;
- $\mathtt{M} \overset{\mathtt{op}_i}{\to} \mathtt{N}_i$ for a unique operation $\mathtt{op}^{+}$ and a unique family $\{\mathtt{N}_i\}_{i=1}^{\mathrm{ar}(\mathtt{op})}$ of closed terms of type τ; and
- $\mathtt{M} \downarrow_{\mathtt{op}}$ for a unique operation $\mathtt{op}^0$.

(II) Any closed term $\mathtt{M}$ of type τ satisfies just one of the following:

- $\mathtt{M} \Rightarrow \mathtt{V}$ for unique (closed) value $\mathtt{V}$ of type τ;
- $\mathtt{M} \overset{\mathtt{op}_i}{\Rightarrow} \mathtt{N}_i$ for a unique operation $\mathtt{op}^{+}$ and a unique family $\{\mathtt{N}_i\}_{i=1}^{\mathrm{ar}(\mathtt{op})}$ of closed terms of type τ;
- $\mathtt{M} \Downarrow_{\mathtt{op}}$ for a unique operation $\mathtt{op}^0$; and
- $\mathtt{M} \Uparrow$. $\square$

Additionally the notion of *effect values* is introduced with the intention of defining a big step operational semantics. The notion is formalized using *continuous* Σ*-algebras*. A continuous Σ-algebra $\mathcal{A}$ is an ω-cppo (i.e. ω-cpo with the least element Ω), with each operation $\mathtt{op} \in \Sigma$ identified as a continuous function from the

$\mathrm{ar}(\mathtt{op})$-fold product of $\mathcal{A}$ to $\mathcal{A}$. Especially for a set X, there exists the free continuous Σ-algebra $CT_\Sigma(X)$ over it.

Let $\mathbf{Val}_\tau$ be the set of values of type τ. For a closed term $\mathtt{M}$ of type τ, its effect value $|\mathtt{M}|$ is defined as a limit of its "finite approximations" in the free continuous Σ-algebra $CT_\Sigma(\mathbf{Val}_\tau)$ over $\mathbf{Val}_\tau$. We refer readers to [25] for the precise definition, but intuitively an effect value $|\mathtt{M}| \in CT_\Sigma(\mathbf{Val}_\tau)$ can be understood as a Σ-branching (possibly infinite) tree over the set $\mathbf{Val}_\tau \cup \{\Omega\}$ that is equal to:

$$\begin{array}{cl}
\mathtt{V} & \text{if } \mathtt{M} \Rightarrow \mathtt{V} \\
\begin{array}{c} \mathtt{op}^{+} \\ / \quad \diagdown \\ |\mathtt{N}_1| \cdots |\mathtt{N}_{\mathrm{ar}(\mathtt{op})}| \end{array} & \text{if } \mathtt{M} \overset{\mathtt{op}_i}{\Rightarrow} \mathtt{N}_i \text{ for each } i \in 1, \ldots, \mathrm{ar}(\mathtt{op}) \\
\mathtt{op}^{0} & \text{if } \mathtt{M} \Downarrow_{\mathtt{op}} \\
\Omega & \text{if } \mathtt{M} \Uparrow \\
|\mathtt{N}| & \text{if } \mathtt{M} \to \mathtt{N} \enspace .
\end{array}$$

3. Algebraic Operations on Monads

Our framework, as well as the mGoI framework [16], accommodates algebraic effects in a uniform way: it exploits their categorical abstraction by monads and algebraic operations [23, 25]. Namely our framework is parametrized by a monad T on $\mathbf{Set}$, where $\mathbf{Set}$ is the category of sets and functions.

3.1 Monads

A monad T on $\mathbf{Set}$ induces the Kleisli category $\mathbf{Set}_T$ equipped with: the finite coproduct structure $(+, \emptyset)$, the countable coproduct structure inherited from $\mathbf{Set}$ (see e.g. [21]), and the premonoidal structure [27]. In order to accommodate recursion, our framework requires these structures to satisfy some domain-theoretic properties stated in Requirement 3.1 below.

Requirement 3.1. Throughout the paper a monad T on $\mathbf{Set}$ is required to satisfy the followings.

- The Kleisli category $\mathbf{Set}_T$ is $\mathbf{Cppo}$-enriched, i.e.
  - every homset $\mathbf{Set}_T(X, Y)$ is an ω-cppo with the least element $\bot$; and
  - compositions $\circ_T$ of $\mathbf{Set}_T$ are continuous.
- Compositions $\circ_T$ are additionally strict in the restricted form: for any $\mathbf{Set}_T$-arrow $f\colon X \to_T Y$ and $\mathbf{Set}$-arrow $g\colon Y \to Z$ it holds that $f \circ_T \bot = \bot$ and $\bot \circ_T g^* = \bot$. Here $(-)^*$, the Kleisli inclusion from $\mathbf{Set}$ to $\mathbf{Set}_T$, assigns a $\mathbf{Set}_T$-arrow $A \to_T B$ to each $\mathbf{Set}$-arrow $A \to B$ by post-composing the unit of T.
- The finite coproduct structure $(+, \emptyset)$ of $\mathbf{Set}_T$ is $\mathbf{Cppo}$-enriched, i.e. cotuplings $[-,-]_T$ are continuous.
- The premonoidal structure $\otimes$ of $\mathbf{Set}_T$ is continuous and strict: namely, for any set X, the maps $X \otimes (-)$ and $(-) \otimes X$ on homsets of $\mathbf{Set}_T$ are continuous and strict.

The first three conditions of this requirement are precisely what are given in [16, Lemma 4.3] as a sufficient condition for monads that can be used in the mGoI framework. It is also shown in [16, Lemma 4.3] that these conditions ensures the existence of a trace operator tr of the monoidal category $(\mathbf{Set}_T, 0, +)$: the operator tr is defined exploiting the $\mathbf{Cppo}$-enrichment of $\mathbf{Set}_T$. Here we additionally require the last condition in order to accommodate recursion. Requirement 3.1 not only supports the development of a component calculus but also ensures its desired domain-theoretic properties.

A monad T, as a parameter of our framework, models computational effects in a uniform way as proposed in [23]; we can instantiate it to model various effects.

Example 3.2. Here are our leading examples of monads that satisfy Requirement 3.1. Note that all the monads T given below are equipped with partiality: it is often obtained by adding $1 = \{*\}$ and induces an ω-cppo structure of each set TX. Hence a $\mathbf{Cppo}$-enrichment of the category $\mathbf{Set}_T$ is given in the pointwise manner.

- The *exception* monad $\mathcal{E}X = 1 + E + X$ for a set E.
- The *powerset* monad $\mathcal{P}X = \{A \subseteq X\}$.
- The *subdistribution* monad $\mathcal{D}X = \{d\colon X \to [0,1] \mid \sum_{x \in X} d(x) \le 1\}$.
- A *global state* monad $\mathcal{S}X = (1 + X \times S)^S$ for a set S.
- A *writer* monad $TX = 1 + M \times X$ for a monoid M.
- An *I/O* monad $TA = \mu X.\,(1 + O \times X + X^I + A)$ for sets I and O. By regarding sets I and O as ω-cpo's with discrete orders, for any ω-cppo X, $F_A X := (1 + O \times X + X^I + A)$ becomes an ω-cppo with the least element $* \in 1$. Hence F_A is an endofunctor on $\mathbf{Cppo}$. The monad sends a set A to the carrier of a final F_A-coalgebra in $\mathbf{Cppo}$.

3.2 Algebraic Operations

Following [25, 26] operations in Σ are modeled by *algebraic operations* on a monad T that serve as interfaces of algebraic effects. We use one equivalent definition of algebraic operations investigated in [26], with finite arities; here X^n denotes a n-fold product of a set X, for $n \in \mathbb{N}$.

Definition 3.3 (algebraic operations on T [26])**.** Let $(\times, 1, \Rightarrow)$ be the cartesian closed structure of $\mathbf{Set}$ and $n \in \mathbb{N}$ be a natural number. For a monad T on $\mathbf{Set}$, a family $\{\alpha_{A,B}\colon (A \Rightarrow TB)^n \to (A \Rightarrow TB)\}_{A \in \mathbf{Set}^{\mathrm{op}}, B \in \mathbf{Set}_T}$ of $\mathbf{Set}$-arrows is an *n-ary algebraic operation on T* if it is natural in $A \in \mathbf{Set}^{\mathrm{op}}$ and $B \in \mathbf{Set}_T$.

The following examples are algebraic operations on some of the monads listed in Example 3.2. They are used in our framework to model operations shown in Example 2.1.

Example 3.4. Here are algebraic operations on some of the monads listed in Example 3.2.

- For a set E and each element $e \in E$, a 0-ary algebraic operation raise_e on the exception monad $\mathcal{E}$ is equivalently given by the family $\{A \to 1 + E + A\}_{A \in \mathbf{Set}}$ of constant functions that always returns e.
- A 2-ary algebraic operation $\oplus$ on the powerset monad $\mathcal{P}$ performs *nondeterministic choice*. For two functions $f, g\colon A \to \mathcal{P}B$ it takes pointwise unions: $(f \oplus g)(a) = f(a) \cup g(a)$.
- For any $p \in [0,1]$, a 2-ary algebraic operation $\oplus_p$ on the subdistribution monad $\mathcal{D}$ performs *probabilistic choice*. For two functions $f, g\colon A \to \mathcal{D}B$ it superposes distributions in a pointwise manner: $(f \oplus_p g)(a)(b) = p \times f(a)(b) + (1-p) \times g(a)(b)$.
- Let *Val* be a finite set and *Loc* be a set. For their elements $v \in \mathit{Val}$ and $l \in \mathit{Loc}$, a $|\mathit{Val}|$-ary algebraic operation lookup_l and a 1-ary algebraic operation $\mathit{update}_{l,v}$ on the global state monad $\mathcal{S}X = (1 + X \times \mathit{Val}^{\mathit{Loc}})^{\mathit{Val}^{\mathit{Loc}}}$ perform *actions on global states*.

4. Component Calculus in the mGoI Framework

In this section we recall the component calculus over *transducers* that is given in the mGoI framework [16].

Transducers, to which our framework translates terms, are precisely called T-transducers: they are "effectful" extension of (sequential) transducers or Mealy machines.

Definition 4.1 (T-transducers [16, Definition 4.1])**.** For sets A and B, a *T-transducer (X, c, x) from A to B* consists of a set X, a function $c\colon X \times A \to T(X \times B)$ and an element $x \in X$.

We write $(X, c, x) \colon A \multimap B$ when a T-transducer (X, c, x) is from A to B. A T-transducer $(X, c, x) \colon A \multimap B$ can be understood as an "effectful" transition function c with input A, output B, a state space X and an initial state $x \in X$. On this intuition we depict it as in Figure 3.

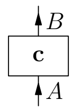


Figure 3: A T-transducer $(X, c, x) \colon A \multimap B$

The component calculus over T-transducers provides some primitive T-transducers and a set of operators on T-transducers.

4.1 Primitive T-transducers

Each $\mathbf{Set}_T$-arrow $f \colon A \to_T B$ (i.e. function $f \colon A \to TB$) can be lifted to a T-transducer $J(f) := (1, f, *) \colon A \multimap B$, where $1 = \{*\}$ is the terminal object of $\mathbf{Set}$ and isomorphisms $1 \times A \overset{\cong}{\Rightarrow} A$ and $1 \times B \overset{\cong}{\Rightarrow} B$ are omitted. The resulting T-transducer Jf performs the same computation as the function f without changing its internal state.

Primitive T-transducers used in our framework are all either in the form $J(f) \colon A \multimap B$ for some $\mathbf{Set}_T$-arrow $f \colon A \to_T B$ or in the form $J(g^*) \colon A \multimap B$ for some $\mathbf{Set}$-arrow $g \colon A \to B$. Here $(-)^*$ is the Kleisli inclusion from $\mathbf{Set}$ to $\mathbf{Set}_T$. For notational simplicity, in depictions of a primitive T-transducer $J(f)$ we simply write its underlying function, e.g. we write f for $J(f)$ and g for $J(g^*)$ (see e.g. Figure 5). We here give the underlying functions of our primitive T-transducers.

One class of underlying functions consists of *retractions* in $\mathbf{Set}$ and $\mathbf{Set}_T$, that are pairs of arrows $f \colon X \lhd Y \colon g$ such that $g \circ f = \mathrm{id}_X$ holds. Namely we use: two chosen bijections in $\mathbf{Set}$

$$u : F\mathbb{N} \cong \mathbb{N} : v \qquad\qquad \phi : \mathbb{N} + \mathbb{N} \cong \mathbb{N} : \psi \tag{1}$$

and four retractions

$$\begin{aligned} \tilde{e}_A &: A \lhd FA : \tilde{e'}_A && \text{(dereliction)} \\ \tilde{d}_A &: FFA \cong FA : \tilde{d'}_A && \text{(digging)} \\ \tilde{c}_A &: FA + FA \cong FA : \tilde{c'}_A && \text{(contraction)} \\ \tilde{w}_A &: \emptyset \lhd FA : \tilde{w'}_A && \text{(weakening)} \end{aligned} \tag{2}$$

where a set FX is defined by $\mathbb{N} \times X$ for each set X. The first three are in $\mathbf{Set}$ (and hence in $\mathbf{Set}_T$) and the last one is in $\mathbf{Set}_T$. Let $A \xrightarrow{\mathrm{inl}} A + B \xleftarrow{\mathrm{inr}} B$ and $A \xrightarrow{\mathrm{inj}_i} \coprod_i A_i$ be injections in $\mathbf{Set}$ and $!_A \colon \emptyset \to A$ be the unique $\mathbf{Set}$-arrow from the initial object $\emptyset$ to A. The retractions in (2) are defined as below.

$$\begin{aligned} \tilde{e}_A &:= \mathrm{inj}_0 & \tilde{e'}_A &:= [\mathrm{id}_A]_{i \in \mathbb{N}} \\ \tilde{d}_A &:= u \times A & \tilde{d'}_A &:= v \times A \\ \tilde{c}_A &:= \phi \times A & \tilde{c'}_A &:= \psi \times A \\ \tilde{w}_A &:= !_{\mathbb{N} \times A} & \tilde{w'}_A &:= \mathrm{tr}^{FA}_{FA,\emptyset}([\mathrm{id}_{FA}, \mathrm{id}_{FA}]^*) \end{aligned}$$

We also use their adaptation as listed below: we take $\mathbb{N}$ as A and compress $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ via the bijection $u : \mathbb{N} \times \mathbb{N} \cong \mathbb{N} : v$ in order to fit the retractions to our use in the translation.

$$\begin{aligned} e &: \mathbb{N} \lhd \mathbb{N} : e' && \text{(dereliction)} \\ d &: \mathbb{N} \times \mathbb{N} \cong \mathbb{N} : d' && \text{(digging)} \\ c &: \mathbb{N} + \mathbb{N} \lhd \mathbb{N} : c' && \text{(contraction)} \\ w &: \emptyset \lhd \mathbb{N} : w' && \text{(weakening)} \end{aligned}$$

Their definition can be concretely expressed as below. We write $\mathtt{g}$ for $\phi \circ \mathrm{inl}_{\mathbb{N},\mathbb{N}}$, $\mathtt{d}$ for $\phi \circ \mathrm{inr}_{\mathbb{N},\mathbb{N}}$ and $\langle -, - \rangle$ for $u(-, -)$.[2]

$$\begin{aligned} e(n) &= \langle 0, n \rangle & c(\mathrm{inl}\langle i, n \rangle) &= \langle \mathtt{g}i, n \rangle \\ e'\langle i, n \rangle &= n & c(\mathrm{inr}\langle i, n \rangle) &= \langle \mathtt{d}i, n \rangle \\ d(i, \langle j, n \rangle) &= \langle \langle i, j \rangle, n \rangle & c'\langle \mathtt{g}i, n \rangle &= \mathrm{inl}\langle i, n \rangle \\ d'\langle \langle i, j \rangle, n \rangle &= (i, \langle j, n \rangle) & c'\langle \mathtt{d}i, n \rangle &= \mathrm{inr}\langle i, n \rangle \\ w &= !^*_{\mathbb{N}} & w' &= \mathrm{tr}^{\mathbb{N}}_{\mathbb{N},\emptyset}([\mathrm{id}_{\mathbb{N}}, \mathrm{id}_{\mathbb{N}}]^*) \end{aligned}$$

[2] $\mathtt{g}$ is for *left* and $\mathtt{d}$ is for *right*, in French. See e.g. [20, 24].

Another class of underlying functions includes the following functions:

$$\begin{aligned} &\mathrm{k}_n : \mathbb{N} \to \mathbb{N} \qquad\qquad \mathrm{sum} \colon \mathbb{N} + \mathbb{N} + \mathbb{N} \to \mathbb{N} + \mathbb{N} + \mathbb{N} \\ &\mathrm{h} := (\phi + \mathbb{N}) \circ (\mathbb{N} + \sigma) \circ (\psi + \mathbb{N}) \circ \sigma \\ &\qquad \circ (\phi + \mathbb{N}) \circ (\mathbb{N} + \sigma) \circ (\psi + \mathbb{N}) \\ &\quad : \mathbb{N} + \mathbb{N} \to \mathbb{N} + \mathbb{N} \end{aligned}$$

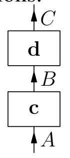
where $\sigma \colon \mathbb{N} + \mathbb{N} \overset{\cong}{\Rightarrow} \mathbb{N} + \mathbb{N}$ is a swapping isomorphism. In the translation, k_n is used for the constant term n; sum is for the arithmetic operation $+$; and h serves as a CPS-like construct that "forces" call-by-value evaluation. They concretely act on natural numbers as below.

$$\begin{aligned} \mathrm{sum}(\mathrm{inj}_0(n)) &= \mathrm{inj}_2(n) & \mathrm{h}(\mathrm{inl}(\mathtt{g}n)) &= \mathrm{inl}(\mathtt{dg}n) \\ \mathrm{sum}(\mathrm{inj}_2\langle i, n \rangle) &= \mathrm{inj}_1\langle \langle i, n \rangle, n \rangle & \mathrm{h}(\mathrm{inl}(\mathtt{dg}n)) &= \mathrm{inl}(\mathtt{g}n) \\ \mathrm{sum}(\mathrm{inj}_1\langle \langle i, n \rangle, m \rangle) &= \mathrm{inj}_0\langle i, n + m \rangle & \mathrm{h}(\mathrm{inl}(\mathtt{dd}n)) &= \mathrm{inr}(n) \\ \mathrm{k}_n\langle i, m \rangle &= \langle i, n \rangle & \mathrm{h}(\mathrm{inr}(n)) &= \mathrm{inl}(\mathtt{dd}n) \end{aligned}$$

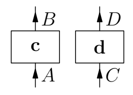
4.2 Operators on T-transducers

The mGoI framework gives the following operators on T-transducers: (a) sequential composition $\circ$; (b) binary parallel composition $\boxplus$; (c) the countable copy operator F; (d) the trace (or feedback) operator Tr; (e) binary application $\bullet$; and (f) the operator $\overline{\alpha}$ for each algebraic operation α on T.

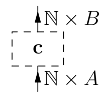
Figure 4 shows depictions of how these operators act on T-transducers. Here we only give intuitive descriptions, for the lack of space; the reader is referred to [16, Section 4.2] for their precise definitions.



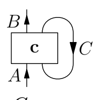
(a) Sequential Composition $\circ$



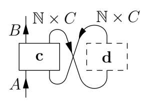
(b) Binary Parallel Composition $\boxplus$



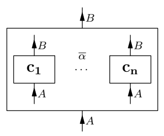
(c) $F(X, c, x) \colon \mathbb{N} \times A \multimap \mathbb{N} \times B$ (for $c \colon A \multimap B$)



(d) $\mathrm{Tr}^C_{A,B}(X, c, x) \colon A \multimap B$



(e) Binary Application $\bullet$



(f) $\overline{\alpha}\{(X_i, c_i, x_i)\}_{i \in \mathbf{n}} \colon A \multimap B$

Figure 4: Operators on T-transducers

Sequential Composition $\circ$ Sequential composition $(Y, d, y) \circ (X, c, x) \colon A \multimap C$ of two T-transducers $(X, c, x) \colon A \multimap B$ and $(Y, d, y) \colon B \multimap C$ has a state space $X \times Y$ and an initial state $\langle x, y \rangle$. It runs first (X, c, x) and then (Y, d, y) by passing output of the former to input of the latter.

Binary Parallel Composition $\boxplus$ Binary parallel composition $(X, c, x) \boxplus (Y, d, y) \colon A + C \multimap B + D$ of two T-transducers $(X, c, x) \colon A \multimap B$ and $(Y, d, y) \colon C \multimap D$ has also a state space $X \times Y$ and an initial state $\langle x, y \rangle$. It runs either (X, c, x) or (Y, d, y) according to input: namely it runs (X, c, x) if input is in A and (Y, d, y) if in C.

Countable Copy Operator F A T-transducer $F(X,c,x)\colon \mathbb{N}\times A \rightarrowtriangle \mathbb{N}\times B$ can be understood as countably many copies of a T-transducer $(X,c,x)\colon A \rightarrowtriangle B$. Given input $\langle i,a\rangle \in \mathbb{N}\times A$, it runs the i-th copy of (X,c,x) with input a; and it outputs $\langle i,b\rangle \in \mathbb{N}\times B$ if the i-th copy of (X,c,x) outputs b.

Trace Operator Tr Trace $\mathrm{Tr}^{C}_{A,B}(X,c,x)\colon A \rightarrow B$ of a T-transducer $(X,c,x)\colon A+C \rightarrowtriangle B+C$ has the same state space and initial state as (X,c,x). Given input in A, it runs (X,c,x) repeatedly until output in B is generated, by passing output in C to input. Its "effectful" transition function is defined using the trace operator tr of the category $\mathbf{Set}_T$.

Binary Application $\bullet$ Using these four operators above, binary application $(X,c,x)\bullet(Y,d,y)\colon A \rightarrowtriangle B$ of two T-transducers $(X,c,x)\colon A+\mathbb{N}\times C \rightarrowtriangle B+\mathbb{N}\times C$ and $(Y,d,y)\colon C \rightarrowtriangle C$ is defined to be

$$\mathrm{Tr}^{\mathbb{N}\times C}_{A,B}\big((J(\mathrm{id}^*_B)\boxplus F(Y,d,y))\circ(X,c,x)\big) \ .$$

Binary application $\bullet$ is an adaptation of the operator $\bullet$ defined in [16]. It is used to translate function application.

Lifted Algebraic Operations $\overline{\alpha}$ Finally for an n-ary algebraic operation α on T and a family $\{(X_i,c_i,x_i)\colon A \rightarrowtriangle B\}_{i\in\mathbf{n}}$ of T-transducers, a T-transducer $\overline{\alpha}\{(X_i,c_i,x_i)\}_{i\in\mathbf{n}}\colon A \rightarrowtriangle B$ has a state space $1+X_1+\cdots+X_n$ and an initial state $*\in 1$. Here the set $\mathbf{n}$ is the n-fold coproduct of 1.

The operator $\overline{\alpha}$ introduces a fresh initial state $*\in 1$ and effectful transitions from the state $*$ to each states in $\{x_i\}_{i\in\mathbf{n}}$ that are initial states of T-transducers $\{(X_i,c_i,x_i)\}_{i\in\mathbf{n}}$, respectively. After making an effectful transition from its initial state $*$ to a state x_i, the T-transducer $\overline{\alpha}\{(X_i,c_i,x_i)\}_{i\in\mathbf{n}}$ sticks to its choice and behaves in the same way as (X_i,c_i,x_i). In other words a T-transducer exploits its internal states to memorize the result of its effectful transitions.

4.3 The "Category" of Transducers

We have introduced the component calculus over T-transducers that is used in the original mGoI framework; one natural question is what axioms it satisfies. An answer given in [16], following the previous observations in [?], is a categorical one via the *behavioral equivalence*.

Definition 4.2 (homomorphism between T-transducers [16, Definition 5.1])**.** For two T-transducers $(X,c,x),(Y,d,y)\colon A \rightarrowtriangle B$, a function $h\colon X\rightarrow Y$ is a *homomorphism from* (X,c,x) *to* (Y,d,y) if $(h^*\otimes B)\circ_T c = d\circ_T (h^*\otimes A)$ holds in $\mathbf{Set}_T$ (equivalently $T(h\times B)\circ c = d\circ(h\times A)$ holds in $\mathbf{Set}$) and $h(x)=y$ holds.

Definition 4.3 (behavioral equivalence [16, Definition 5.2])**.** Two T-transducers $(X,c,x),(Y,d,y)\colon A \rightarrowtriangle B$ are *behavioral equivalent* if there exists a T-transducer $(Z,e,z)\colon A \rightarrowtriangle B$ and two homomorphisms, from (X,c,x) to (Z,e,z) and from (Y,d,y) to (Z,e,z).

The behavioral equivalence enables us to abstract away from state spaces of T-transducers. For example two T-transducers $(X,c,x)\colon A \rightarrowtriangle B$ and $J(\mathrm{id}^*_B)\circ(X,c,x)\colon A \rightarrowtriangle B$ have different state spaces (namely X and $X\times 1$) but their transition functions are identified after composing an obvious isomorphism $X\cong X\times 1$. By choosing the isomorphism as a homomorphism between the two T-transducers we can identify them via the behavioral equivalence.

All the operators $\circ$, $\boxplus$, Tr and F as well as $\overline{\alpha}$ respect the behavioral equivalence. The axioms satisfied by these operators can be described categorically; what follows is a summary of the facts shown in [16].

- The category $\mathbf{Res}(T)$, defined by
  - objects: sets
  - arrows: *resumptions*, that are equivalence classes of T-transducers modulo the behavioral equivalence, with identities given by $J(\mathrm{id}^*)$ and compositions by $\circ$

  is indeed a category and has a traced symmetric monoidal structure $(\boxplus,\emptyset,\mathrm{Tr})$.
- The operator F, defined for sets by $FX := \mathbb{N}\times X$, is a traced symmetric monoidal functor on $\mathbf{Res}(T)$. This fact enables us to identify a T-transducer $F(X,c,x)\colon F(A+C)\rightarrowtriangle F(B+D)$ with $F(X,c,x)\colon FA+FC \rightarrowtriangle FB+FD$, as done e.g. in Figure 5 & 7.
- Moreover the data $(\mathbf{Res}(T),F,\mathbb{N})$, together with primitive T-transducers lifted from the retractions in (1) and (2), induces a GoI situation [2]. This means the primitive T-transducers lifted from the left-hand side functions of retractions in (2) satisfies monoidal naturality.
- [16, Theorem 5.3] Operators $\overline{\alpha}$ are natural and Tr distributes over them up to the behavioral equivalence.

Figures in this section can be therefore seen as string diagrams in $\mathbf{Res}(T)$.

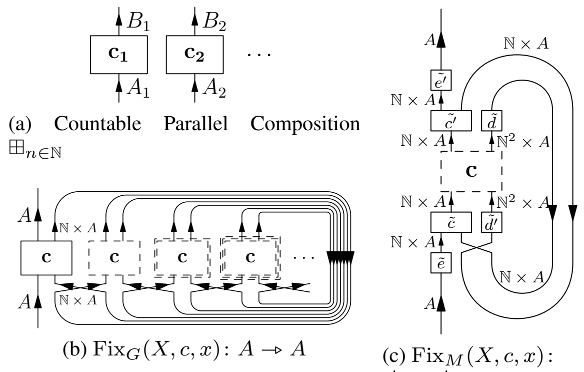
It is also possible to think of a "category" $\mathbf{Trans}(T)$ of T-transducers (without quotienting modulo the behavioral equivalence); however this is more like a bicategory since axioms like associativity of $\circ$ are satisfied only up to isomorphisms.

5. Extending the Component Calculus

In the previous section we have recalled the component calculus over T-transducers from the mGoI framework; now in this section it is extended so that our framework can accommodate recursion.

In the mGoI framework, binary application $\bullet$ of T-transducers is used in translating function application to T-transducers. Our goal here is to introduce a new operator that can be used in translating recursion as infinitely many self-applications of some function. Hence the new operator is designed to give infinitely many "self-binary-applications" of a given T-transducer.

We extend the component calculus in three steps by introducing first countable parallel composition $\boxplus_{n\in\mathbb{N}}$, second the Girard style fixed point operator Fix_G and finally the Mackie style fixed point operator Fix_M. Figure 5 shows depictions of how these new operators act on T-transducers.



(a) Countable Parallel Composition $\boxplus_{n\in\mathbb{N}}$

(b) $\mathrm{Fix}_G(X,c,x)\colon A \rightarrowtriangle A$

(c) $\mathrm{Fix}_M(X,c,x)\colon A \rightarrowtriangle A$

Figure 5: New Operators on T-transducers

5.1 Countable Parallel Composition $\boxplus_{n\in\mathbb{N}}$

As a first step we extend binary parallel composition $\boxplus$ to the countable one $\boxplus_{n\in\mathbb{N}}$; this is straightforward, and possible because $\mathbf{Set}$ has countable coproducts and $\mathbf{Set}_T$ inherits them. Countable parallel composition $\boxplus_{n\in\mathbb{N}}\{(X_n,c_n,x_n)\}\colon \coprod_{n\in\mathbb{N}}A_n \rightarrowtriangle \coprod_{n\in\mathbb{N}}B_n$

of a family $\{(X_n, c_n, x_n)\colon A_n \twoheadrightarrow B_n\}_{n\in\mathbb{N}}$ of T-transducers has a state space $\prod_{n\in\mathbb{N}} X_n$, whose element is an infinite list $\langle y_1, y_2, \ldots\rangle$, and has an initial state $\langle x_1, x_2, \ldots\rangle$.

Countable parallel composition $\boxplus_{n\in\mathbb{N}}$ can be seen as a generalization of the countable copy operator F. Namely for any T-transducer $(X, c, x)\colon A \twoheadrightarrow B$, a T-transducer $F(X, c, x)$ can be identified with a T-transducer $\boxplus_{n\in\mathbb{N}}\{(X, c, x)\}$. Here the former's input/output $\mathbb{N} \times C$ is identified with a $\mathbb{N}$-fold countable coproduct of C and the former's state space $X^{\mathbb{N}}$ is with a $\mathbb{N}$-fold countable product of X.

5.2 Girard Style Fixed Point Operator Fix_G

On top of the component calculus, i.e. operators on T-transducers described so far, we define the Girard style fixed point operator Fix_G on T-transducers.

Definition 5.1 (Girard style fixed point operator Fix_G)**.** For a T-transducer $(X, c, x)\colon A + \mathbb{N} \times A \twoheadrightarrow A + \mathbb{N} \times A$, a T-transducer $\mathrm{Fix}_G(X, c, x)\colon A \twoheadrightarrow A$ is defined by

$$\mathrm{Fix}_G(X, c, x) := \mathrm{Tr}^N_{A,A}(\boxplus_{n\in\mathbb{N}}\{F^n(X, c, x)\}_{n\in\mathbb{N}} \circ J(\mathit{swap}^*))$$

where N is the set defined to be

$$\begin{aligned}(\mathbb{N} \times A + \mathbb{N} \times A) &+ (\mathbb{N} \times \mathbb{N} \times A + \mathbb{N} \times \mathbb{N} \times A)\\ &+ (\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times A + \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times A)\\ &+ \cdots\end{aligned}$$

and swap is the function defined to be $\mathrm{id}_A + \coprod_{n\in\mathbb{N}} \sigma\colon N \to N$ using swapping isomorphisms $\{\sigma\colon R + S \xrightarrow{\cong} S + R\}_{R,S\in\mathbf{Set}}$ in $\mathbf{Set}$.

The following proposition shows that the operator Fix_G indeed gives what serves as infinitely many "self-binary-applications" of a transducer, as we intended. Namely the operator Fix_G gives a fixed point with respect to binary application $\bullet$: that is why we call Fix_G a "fixed point" operator.

Proposition 5.2. For any T-transducer $(X, c, x)\colon A + \mathbb{N} \times A \twoheadrightarrow A + \mathbb{N} \times A$, a T-transducer $\mathrm{Fix}_G(X, c, x)\colon A \twoheadrightarrow A$ satisfies the behavioral equivalence

$$(X, c, x) \bullet \mathrm{Fix}_G(X, c, x) \;\simeq\; \mathrm{Fix}_G(X, c, x) \;.$$

Proof. The behavioral equivalence follows from the equational properties (up to behavioral equivalence) listed in Section 4.3, of the component calculus. Especially we take advantage of trace axioms [14, 18] satisfied by the trace operator Tr. □

5.3 Induced ω-cpo Structure on Transducers

The operator Fix_G gives not only a fixed point with respect to binary application $\bullet$ but also a least fixed point with respect to an ω-cpo structure on T-transducers, that is induced by the $\mathbf{Cppo}$-enrichment of the category $\mathbf{Set}_T$.

As guaranteed by Requirement 3.1, each homset of $\mathbf{Set}_T$ has an ω-cppo structure $(\mathbf{Set}_T(A, B), \sqsubseteq, \bot)$. This structure induces an ω-cpo structure $(\mathbf{Trans}(T)(A, B), \trianglelefteq)$ in the following way, where $\mathbf{Trans}(T)(A, B)$ denotes a set of T-transducers from A to B.

- A relation $\trianglelefteq$ on T-transducers, defined by
$$(X, c, x) \trianglelefteq (Y, d, y) \overset{\text{def.}}{\iff} X = Y \wedge x = y \wedge c \sqsubseteq d$$
for two T-transducers $(X, c, x), (Y, d, y)\colon A \twoheadrightarrow B$, is a partial order.
- For an ω-chain $(X, c_1, x) \trianglelefteq (X, c_2, x) \trianglelefteq \cdots$ of T-transducers from A to B, its supremum $\sup_{i\in\omega}(X, c_i, x)\colon A \twoheadrightarrow B$ is given by $(X, \sup_{i\in\omega} c_i, x)$.
- Additionally a T-transducer $(X, \bot, x)\colon A \twoheadrightarrow B$ for each least element $\bot \in \mathbf{Set}_T(X \times A, X \times B)$ gives a minimal element.

This partial order $\trianglelefteq$ is quite "raw": two T-transducers can be ordered only if they have the same state spaces and the same initial states. The order $\trianglelefteq$ forces us to remain aware of state spaces, while the behavioral equivalence $\simeq$ (see Definition 4.3) enables us to abstract away state spaces and to perform equational reasoning up to behavioral equivalence. To our knowledge there seems no way to relate $\trianglelefteq$ to the behavioral equivalence $\simeq$. In spite of this inconvenience, the order $\trianglelefteq$ has enough power to support the proof of adequacy.

The conditions stated in Requirement 3.1 imply the following domain-theoretic properties of our component calculus with respect to the order $\trianglelefteq$.

Lemma 5.3. Operators of the component calculus satisfy the following, up to (not behavioral equivalence but) the exact equality $=$.

	continuity	strictness
sequential composition $\circ$	✓	✓$_{\star_1}$
binary parallel composition $\boxplus$	✓	✓$_{\star_2}$
countable copy operator F	✓	✓
trace operator Tr	✓	✓
binary application $\bullet$	✓	✓$_{\star_3}$
lifted algebraic operation $\overline{\alpha}$	✓	$\times$
countable parallel composition $\boxplus_{n\in\mathbb{N}}$	✓	✓$_{\star_2}$
Girard style fixed point operator Fix_G	✓	✓

$\star_1$ In the restricted form: $(Y, d, y) \circ (X, \bot, x) = (X \times Y, \bot, \langle x, y\rangle)$ and $(X, \bot, x) \circ (Z, e^*, z) = (Z \times X, \bot, \langle z, x\rangle)$, for any T-transducers (X, c, x) and (Y, d, y), and for any T-transducer (Z, e^*, z) whose transition function is lifted from a $\mathbf{Set}$-arrow e.

$\star_2$ If all arguments have $\bot$ as transition functions.

$\star_3$ In the restricted form: $(X, \bot, x) \bullet (Y, d, y) = (Z, \bot, z)$ for any T-transducer (Y, d, y) and any T-transducer $(X, \bot, x)$ whose transition function is $\bot$, where Z and z is respectively the state space and the initial state of $(X, \bot, x) \bullet (Y, d, y)$.

Proof. The properties shown in the table are consequences of the conditions stated in Requirement 3.1. Note that the conditions imply the facts in the category $\mathbf{Set}_T$ listed below. Once we observe them the properties of the component calculus can be easily confirmed by definitions of operators.

- Continuity of countable cotuplings $[\{-\}_{n\in\mathbb{N}}]_T$ is inherited by that of (finite) cotuplings $[-, -]_T$.
- Both finite and countable cotuplings are strict in the sense of $[\bot, \bot]_T = \bot$ and $[\{\bot\}_{n\in\mathbb{N}}]_T = \bot$. It is because of the restricted strictness of composition $\circ_T$.
- The trace operator tr is continuous and strict by its definition that takes advantage of $\mathbf{Cppo}$-enrichment of $\mathbf{Set}_T$.
- Continuity of algebraic operations on T follows from continuity of composition $\circ_T$ and cotuplings $[-, -]_T$.

The last fact can be confirmed using the bijective correspondence of an n-ary algebraic operation α on T and a $\mathbf{Set}_T$-arrow (called *generic effects*) $\beta\colon 1 \to_T \mathbf{n}$, studied in [26]. Namely for a family $\{f_i\colon A \to_T B\}_{i\in\mathbf{n}}$ of $\mathbf{Set}_T$-arrows, a $\mathbf{Set}_T$-arrow $\alpha\{f_i\}_{i\in\mathbf{N}}\colon A \to_T B$ can be equivalently given by

$$A \xrightarrow{\beta\otimes A}_T \mathbf{n} \times A \xrightarrow{\cong}_T A + \cdots + A \xrightarrow{[\{f_i\}_{i\in\mathbf{n}}]_T}_T B$$

using the corresponding generic effect β. □

Based on these properties the operator Fix_G is characterized as a supremum (or a least fixed point) of its finite approximants.

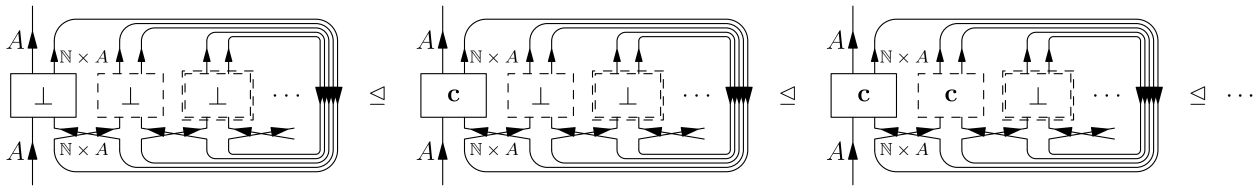


Figure 6: The ω-chain of Finite Approximants $\mathrm{Fix}_G^{(i)}(X, c, x)$

Definition 5.4 (finite approximant $\mathrm{Fix}_G^{(i)}$). For a T-transducer $(X, c, x)\colon A + \mathbb{N} \times A \multimap A + \mathbb{N} \times A$ and $i \in \omega$, a T-transducer $\mathrm{Fix}_G^{(i)}(X, c, x)\colon A \multimap A$ is defined by

$$\mathrm{Fix}_G^{(i)}(X, c, x) := \mathrm{Tr}^N_{A,A}(\boxplus_{n\in\mathbb{N}}\{F^n(\mathbf{c}_n^{(i)})\}_{n\in\mathbb{N}} \circ J(swap^*))$$

where N and $swap$ are defined as in Definition 5.1; and $\mathbf{c}_n^{(i)}\colon A + \mathbb{N} \times A \multimap A + \mathbb{N} \times A$ is the T-transducer defined by (X, c, x) if $n < i$ and $(X, \perp, x)$ otherwise, for each $i \in \omega$ and $n \in \mathbb{N}$.

Proposition 5.5. For any T-transducer $(X, c, x)\colon A + \mathbb{N} \times A \multimap A + \mathbb{N}\times A$, a family $\{\mathrm{Fix}_G^{(i)}(X, c, x)\colon A \multimap A\}_{i\in\omega}$ of T-transducers forms an ω-chain

$$\mathrm{Fix}_G^{(0)}(X, c, x) \trianglelefteq \mathrm{Fix}_G^{(1)}(X, c, x) \trianglelefteq \cdots$$

as depicted in Figure 6. Moreover a T-transducer $\mathrm{Fix}_G(X, c, x)\colon A \multimap A$ satisfies the behavioral equivalence

$$\mathrm{Fix}_G(X, c, x) \;\simeq\; \sup_{i\in\omega}(\mathrm{Fix}_G^{(i)}(X, c, x)) \;. \qquad \square$$

5.4 Mackie Style Fixed Point Operator Fix_M

So far, in this section, the Girard style fixed point operator Fix_G has been introduced with the intention of using it in translating recursion. As shown in Proposition 5.2 & 5.5, the operator Fix_G enjoys properties that are essential in the proof of adequacy; especially it is characterized as a supremum of its finite approximants. Therefore by the operator Fix_G recursion can be translated as a supremum of finite approximants. This approach to interpret recursion in GoI is much like the one by Girard [12]; that is why we call Fix_G "Girard style" fixed point operator.

On the other hand there exists another approach to interpret recursion in GoI: that is the one by Mackie [22] where recursion is interpreted by a loop and a single "box" in a proof net. In our setting this approach corresponds to interpreting recursion using an operator that is defined by the single use of the trace operator Tr and the single use of countable copy operator F, in particular without any use of countable parallel composition $\boxplus_{n\in\mathbb{N}}$. The last operator Fix_M on T-transducers, introduced in our framework, is defined in such a way: therefore we call the operator "Mackie style" fixed point operator. See Figure 5 for its depiction.

Definition 5.6 (Mackie style fixed point operator Fix_M). For a T-transducer $(X, c, x)\colon A + \mathbb{N} \times A \multimap A + \mathbb{N} \times A$, a T-transducer $\mathrm{Fix}_M(X, c, x)\colon A \multimap A$ is defined by

$$\begin{aligned}\mathrm{Fix}_M(X, c, x) :=\; & \\ \mathrm{Tr}^{A+A}_{A,A}((J(\tilde{e'}^*) \boxplus J(\mathrm{id}^*_{A+A})) \circ (J(\tilde{c'}^*) \boxplus J(\tilde{d}^*)) \circ F(X, c, x) \\ \circ\, (J(\tilde{c}^*) \boxplus J(\tilde{d'}^*)) \circ (J(\tilde{e}^*) \boxplus J(\sigma^*)))\end{aligned}$$

where σ is a swapping isomorphism $A + A \xrightarrow{\cong} A + A$ in $\mathbf{Set}$.

The next theorem shows that the Girard and Mackie style fixed point operators actually coincide.

Theorem 5.7 (coincidence of two styles of fixed point operator). The following behavioral equivalence

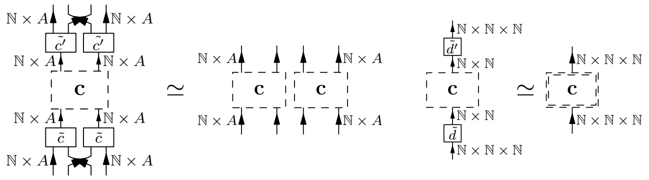
$$\mathrm{Fix}_G(X, c, x) \;\simeq\; \mathrm{Fix}_M(X, c, x)$$

holds for any T-transducer $(X, c, x)\colon A + \mathbb{N} \times A \multimap A + \mathbb{N} \times A$.

Proof. The proof exploits: 1) naturality of the trace operator Tr; and 2) monoidal naturality (in A) of the retractions/isomorphisms

$$\tilde{e}_A : A \lhd FA : \tilde{e'}_A, \quad \tilde{d}_A : FFA \cong FA : \tilde{d'}_A \quad \text{and} \\ \tilde{c}_A : FA + FA \cong FA : \tilde{c'}_A.$$

The latter is exploited e.g. in deriving the following behavioral equivalences of transducers.



$\mathbb{N} \times A$
$\tilde{c'}$
c
$\tilde{c}$
$\simeq$
$\mathbb{N} \times \mathbb{N} \times \mathbb{N}$
$\mathbb{N} \times \mathbb{N}$
$\tilde{d'}$
$\tilde{d}$

We use the above behavioral equivalences for the purpose of translating a bunch of (X, c, x)'s in $\mathrm{Fix}_G(X, c, x)$ into a single $F(X, c, x)$ in $\mathrm{Fix}_M(X, c, x)$. $\square$

6. Translation to Transducers and Its Adequacy

Finally in this section, on top of our extended component calculus, we define a translation of terms of $\mathcal{L}_\Sigma$ to transducers. To achieve adequacy of the translation we exploit the properties of primitives and operators of our component calculus.

6.1 Translation to Transducers

In order to translate a term of our target language $\mathcal{L}_\Sigma$, the first step is to choose an appropriate monad T that satisfies Requirement 3.1 and *supports* the algebraic signature Σ.

Definition 6.1. We say a monad T on $\mathbf{Set}$ *supports* an algebraic signature Σ if, for each operation $\mathtt{op} \in \Sigma$, it has an $\mathrm{ar}(\mathtt{op})$-ary algebraic operation op on it.

For the algebraic signatures listed in Example 2.1, we use the monads listed in Example 3.2. Operations in these algebraic signatures can be modeled by algebraic operations listed in Example 3.4.

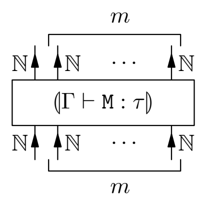
Example 6.2. Here we show how the algebraic signatures listed in Example 2.1 are supported by the monads listed in Example 3.2.

- For a set E, the exception monad $\mathcal{E} = 1 + E + (-)$ supports $\Sigma_{\mathrm{except}} = \{\mathtt{raise}_e \mid e \in E\}$. For each operation $\mathtt{raise}_e \in \Sigma_{\mathrm{except}}$ we can take a 0-ary algebraic operation $raise_e$ on $\mathcal{E}$.
- The powerset monad $\mathcal{P}$ supports Σ_{ndet} where the operation $\sqcup$ is modeled by the 2-ary algebraic operation $\oplus$ on $\mathcal{P}$.
- Similarly the subdistribution monad $\mathcal{D}$ supports Σ_{prob} where each operation $\sqcup_p$ is modeled by the 2-ary algebraic operation $\oplus_p$ on $\mathcal{D}$.

- For a set Loc and a finite set Val, the state monad $\mathcal{S}X = (1 + (-) \times Val^{Loc})^{Val^{Loc}}$ supports $\Sigma_{\mathrm{glstate}}$. Each operation $\mathtt{lookup}_l$ is modeled by the $|Val|$-ary algebraic operation $lookup_l$ and each operation $\mathtt{update}_{l,v}$ is by the 1-ary algebraic operation $update_{l,v}$ on $\mathcal{S}$.

Let T be a monad that supports Σ. We now give a translation $(\!|-|\!)$ of terms of $\mathcal{L}_\Sigma$ to T-transducers. It is given using depictions such as Figure 3–5. Moreover it is an extension of the translation given by the mGoI framework [16].

Definition 6.3 (translation $(\!|-|\!)$)**.** For each type judgment $\Gamma \vdash \mathtt{M} : \tau$ where $\Gamma = \mathtt{x}_1 : \tau_1, \ldots \mathtt{x}_m : \tau_m$, we inductively define a T-transducer

$(\!|\Gamma \vdash \mathtt{M} : \tau|\!) =$ 

$: \coprod_{i=0}^{m} \mathbb{N} \rightharpoonup \coprod_{i=0}^{m} \mathbb{N}$

as in Figure 7, where labels of edges (either $\mathbb{N}$ or $\mathbb{N} \times \mathbb{N}$) are omitted for visibility.

In translation of recursion depicted in Figure 7 we implicitly use the Mackie style fixed point operator Fix_M. Due to Theorem 5.7 we can also use the Girard style fixed point operator Fix_G as depicted in Figure 8.

The Categorical Model Much like the translation of the original mGoI framework in [16], the translation $(\!|-|\!)$ of our framework is backed up by a categorical model (whose definition we do not give here). Our translation $(\!|-|\!)$ can be extracted by a categorical interpretation on the model $\mathbf{Per}_{\Phi'}$ that is the Kleisli category of the strong monad Φ' on the cartesian closed category $\mathbf{Per}$. The model is a modification of the one used in the mGoI framework: the category $\mathbf{Per}$ is the same; and the monad Φ' is modified to reflect the ω-cpo structure of T-transducers. Its construction is done by combination of categorical GoI [2] and realizability techniques, as in [16]. Proposition 5.2 & 5.5 ensures that the Girard style fixed point operator Fix_G indeed yields a (categorical) fixed point operator in the model $\mathbf{Per}_{\Phi'}$.

6.2 Adequacy of Translation $(\!|-|\!)$

Let T be a monad that supports the algebraic signature Σ. The statement of adequacy connects operational semantics reviewed in Section 2.2 and the translation $(\!|-|\!)$ extracted from denotational semantics. In order to give the statement we start with "collecting" the execution results of terms of $\mathcal{L}_\Sigma$ via both T-transducers and effect values. In this section we restrict ourselves to closed terms of base type $\mathtt{nat}$: we simply say a "term" to indicate such a term, and write $(\!|\mathtt{M}|\!)$ for $(\!|\vdash \mathtt{M} : \mathtt{nat}|\!)$.

For a term $\mathtt{M}$, we can observe that running the T-transducer $(\!|\mathtt{M}|\!) \colon \mathbb{N} \rightharpoonup \mathbb{N}$ with input in the form $\mathtt{dd}\langle i, m\rangle$ yields output in the form $\mathtt{dd}\langle i, n\rangle$, where i and m are arbitrary natural numbers and n corresponds to an evaluation result of the term (see Section 4.1 for the notation). Therefore we take the set $T\mathbb{N}$ as the collection of evaluation results, fixing a retraction $\mathrm{enc} : \mathbf{Val}_{\mathtt{nat}} \lhd \mathbb{N} : \mathrm{dec}$ such that

$$\mathrm{enc}(\underline{n}) = \mathtt{dd}\langle 0, n\rangle \qquad \mathrm{dec}(\mathtt{dd}\langle i, n\rangle) = \underline{n} \ .$$

For a term $\mathtt{M}$, let $(X, c, x) \colon \mathbb{N} \rightharpoonup \mathbb{N}$ be a T-transducer $(\!|\mathtt{M}|\!)$. We collect execution results from the T-transducer by taking

$$(\!|\mathtt{M}|\!)^\dagger := ((\pi'_{X,\mathbb{N}})^* \circ_T c)(x, \mathrm{enc}(\underline{m})) \in T\mathbb{N}$$

where $\pi'_{X,\mathbb{N}} \colon X \times \mathbb{N} \to \mathbb{N}$ is the second projection and m denotes a fixed natural number. This procedure $(-)^\dagger$ runs a T-transducer and gathers its output. Additionally $(-)^\dagger$ ignores the resulting internal states that record the branching history of program execution, or the history of effect occurrences during program execution. Indeed we have the equality $(X, c, x)^\dagger = (Y, d, y)^\dagger$ for two behavioral equivalent T-transducers $(X, c, x) \simeq (Y, d, y) \colon A \rightharpoonup B$.

Next we define interpretation of effect values. Recall that effect values for terms are formulated as elements of the free continuous Σ-algebra $CT_\Sigma(\mathbf{Val}_{\mathtt{nat}})$ over the set $\mathbf{Val}_{\mathtt{nat}}$. On the other hand, for each operation $\mathtt{op} \in \Sigma$, the monad T has an $\mathrm{ar}(\mathtt{op})$-ary algebraic operation op on it. This means that the set $T\mathbb{N}$, which is isomorphic to $\mathbf{Set}_T(1, \mathbb{N})$, is a continuous Σ-algebra. Therefore we can lift the function $\mathrm{enc}^* \colon \mathbf{Val}_{\mathtt{nat}} \to T\mathbb{N}$ to a unique morphism $[\![-]\!] \colon CT_\Sigma(\mathbf{Val}_{\mathtt{nat}}) \to T\mathbb{N}$ that is a strict continuous function preserving the operations specified by Σ. When the monad T is equal to the powerset monad $\mathcal{P}$, this interpretation corresponds to forgetting branching structures of effect values (i.e. branching structures of program execution): for example two effect values $(\underline{1} \sqcup \underline{2}) \sqcup \underline{3}$ and $\underline{1} \sqcup (\underline{2} \sqcup \underline{3})$ are identified as a set $\{\mathrm{enc}(\underline{1}), \mathrm{enc}(\underline{2}), \mathrm{enc}(\underline{3})\} \subseteq \mathcal{P}\mathbb{N}$ where $\sqcup \in \Sigma_{\mathrm{ndet}}$ is the nondeterministic choice operation.

Now we can state adequacy of the translation $(\!|-|\!)$.

Theorem 6.4 (adequacy)**.** Any closed term $\mathtt{M}$ of base type $\mathtt{nat}$ satisfies $[\![|\mathtt{M}|]\!] = (\!|\mathtt{M}|\!)^\dagger$.

6.3 Proof of Adequacy

To prove Theorem 6.4 we introduce a language $\overline{\mathcal{L}}_\Sigma$ following [25]. In this language all occurrences of recursion are restricted to finite depth. Namely $\overline{\mathcal{L}}_\Sigma$ is made from the target language $\mathcal{L}_\Sigma$, by replacing the term constructor $\mathtt{rec}$ with $\mathtt{rec}^{(i)}$ for each $i \in \mathbb{N}$ and by adding a constant Ω_τ for each type τ. In the definition of the small step operational semantics, transitions

$$\begin{aligned}
&\mathtt{rec}^{(i+1)}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\,\mathtt{M} \\
&\quad \to (\lambda \mathtt{x} : \sigma.\,\mathtt{M})[\mathtt{rec}^{(i)}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\,\mathtt{M}/\mathtt{f}] \\
&\mathtt{rec}^{(0)}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\,\mathtt{M} \to \lambda \mathtt{x} : \sigma.\,\Omega_\tau
\end{aligned}$$

replace the one for $\mathtt{rec}$ shown in Figure 2. This excludes any infinite sequence $\mathtt{M} \to \mathtt{M}' \to \cdots$ of pure transitions, hence $\mathtt{M} \Uparrow$ no longer appears in the medium step operational semantics. Effect values are defined in the same way as $\mathcal{L}_\Sigma$, except that we define $|\mathtt{E}[\Omega_\tau]|$ to be Ω. On the other hand the translation $(\!|\Gamma \vdash \mathtt{rec}^{(i)}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\,\mathtt{M}|\!)$ is given like Figure 8 where we use the i-th approximant $\mathrm{Fix}_G^{(i)}$ of the Girard style fixed point operator instead of Fix_G.

Theorem 6.4 is a consequence of Lemma 6.5 below that states adequacy of the translation $(\!|-|\!)$ with respect to the language $\overline{\mathcal{L}}_\Sigma$. The proof goes as follows. For a term $\mathtt{M}$ of $\mathcal{L}_\Sigma$ and $i \in \mathbb{N}$, let $\mathtt{M}^{(i)}$ be a term of $\overline{\mathcal{L}}_\Sigma$ obtained from $\mathtt{M}$ by replacing all occurrences of $\mathtt{rec}$ with $\mathtt{rec}^{(i)}$. For effect values we can prove

$$[\![|\mathtt{M}|]\!] = \sup_{i \in \omega} [\![|\mathtt{M}^{(i)}|]\!]$$

in $T\mathbb{N}$ as in [25]. On the other hand Proposition 5.5 ensures the behavioral equivalence

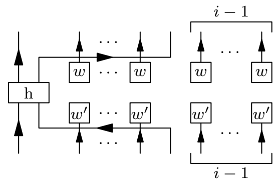
$$(\!|\mathtt{M}|\!) \simeq \sup_{i \in \omega} (\!|\mathtt{M}^{(i)}|\!) \ .$$

Therefore it holds that

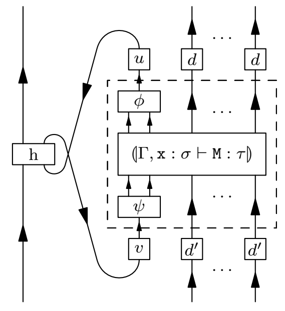
$$[\![|\mathtt{M}|]\!] = \sup_{i \in \omega} [\![|\mathtt{M}^{(i)}|]\!] = \sup_{i \in \omega} ((\!|\mathtt{M}^{(i)}|\!)^\dagger) = (\sup_{i \in \omega} (\!|\mathtt{M}^{(i)}|\!))^\dagger = (\!|\mathtt{M}|\!)^\dagger \ .$$

The third equality is easily confirmed by the definition of $(-)^\dagger$ and continuity of composition $\circ_T$ of $\mathbf{Set}_T$.

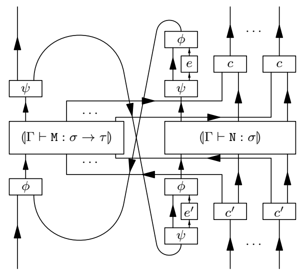
$(\!|\Gamma \vdash \mathtt{x}_i : \tau_i|\!) =$



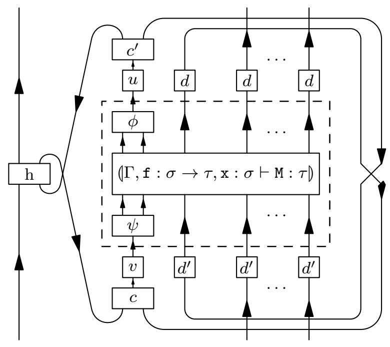
$(\!|\Gamma \vdash \lambda\mathtt{x} : \sigma.\,\mathtt{M} : \sigma \to \tau|\!) =$



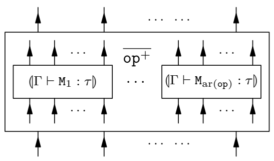
$(\!|\Gamma \vdash \mathtt{M}\ \mathtt{N} : \tau|\!) =$



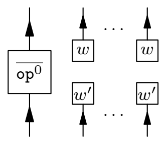
$(\!|\Gamma \vdash \mathtt{rec}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\,\mathtt{M} : \sigma \to \tau|\!) =$



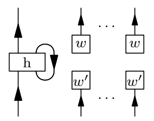
$(\!|\Gamma \vdash \mathtt{op}^+(\mathtt{M}_1, \ldots, \mathtt{M}_{\mathrm{ar(op)}}) : \tau|\!) =$



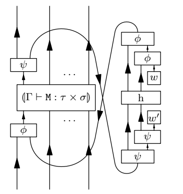
$(\!|\Gamma \vdash \mathtt{op}^0() : \tau|\!) =$



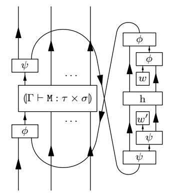
$(\!|\Gamma \vdash \underline{*} : \mathtt{unit}|\!) =$



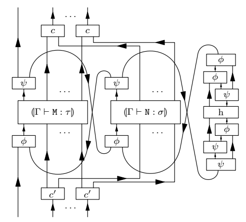
$(\!|\Gamma \vdash \mathtt{fst}(\mathtt{M}) : \tau|\!) =$



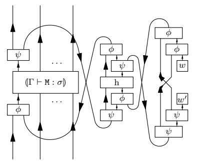
$(\!|\Gamma \vdash \mathtt{snd}(\mathtt{M}) : \sigma|\!) =$



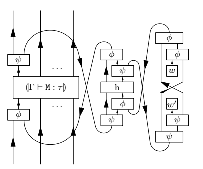
$(\!|\Gamma \vdash \langle \mathtt{M}, \mathtt{N} \rangle : \tau \times \sigma|\!) =$



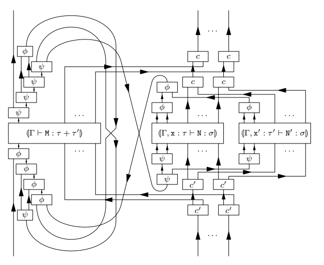
$(\!|\Gamma \vdash \mathtt{inl}_{\tau,\sigma}(\mathtt{M}) : \tau + \sigma|\!) =$



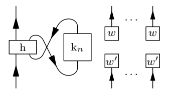
$(\!|\Gamma \vdash \mathtt{inr}_{\tau,\sigma}(\mathtt{M}) : \tau + \sigma|\!) =$



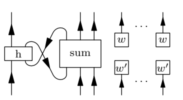
$(\!|\Gamma \vdash \mathtt{case}(\mathtt{M}, \mathtt{x}.\,\mathtt{N}, \mathtt{x}'.\,\mathtt{N}') : \sigma|\!) =$



$(\!|\Gamma \vdash \underline{n} : \mathtt{nat}|\!) =$



$(\!|\Gamma, \mathtt{x} : \mathtt{nat}, \mathtt{y} : \mathtt{nat} \vdash \mathtt{x} + \mathtt{y} : \mathtt{nat}|\!) =$



(if $\mathtt{x} \not\equiv \mathtt{y}$)

$(\!|\Gamma, \mathtt{x} : \mathtt{nat} \vdash \mathtt{x} + \mathtt{x} : \mathtt{nat}|\!) =$

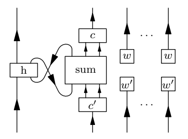


Figure 7: Inductive Definition of the Translation $(\!|-|\!)$

$(\!|\Gamma \vdash \mathtt{rec}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M} : \sigma \to \tau|\!) =$

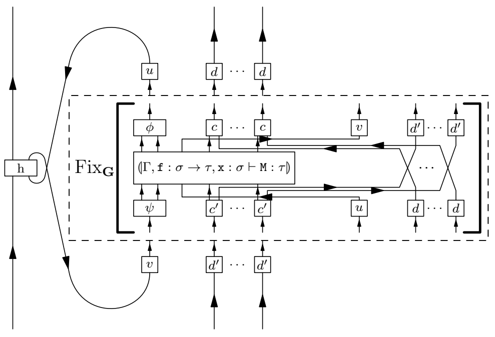


Figure 8: Translation of Recursion by Fix_G

Lemma 6.5 (adequacy with respect to $\overline{\mathcal{L}}_\Sigma$)**.** For the language $\overline{\mathcal{L}}_\Sigma$, any closed term $\mathtt{M}$ of base type $\mathtt{nat}$ satisfies $[\![|\mathtt{M}|]\!] = (\!|\mathtt{M}|\!)^\dagger$.

Proof. The proof utilizes a logical relation between T-transducers and closed terms. Let $\mathbf{Term}_\tau$ be the set of closed terms of type τ. For a binary relation $R \subseteq \mathbf{Trans}(T)(\mathbb{N}, \mathbb{N}) \times \mathbf{Val}_\tau$, a binary relation $\overline{R} \subseteq \mathbf{Trans}(T)(\mathbb{N}, \mathbb{N}) \times \mathbf{Term}_\tau$ is defined to be the least binary relation such that

- $(\mathbf{d}, \mathtt{M}) \in \overline{R}$ if $(\mathbf{c}, \mathtt{M}) \in \overline{R}$ and $\mathbf{c} \simeq \mathbf{d}$
- $(J(\mathrm{h}^*) \bullet' \mathbf{c}, \mathtt{V}) \in \overline{R}$ for any $(\mathbf{c}, \mathtt{V}) \in R$, where h is from Section 4.1
- $(\mathbf{c}, \mathtt{M}) \in \overline{R}$ if $\mathtt{M} \to \mathtt{N}$ and $(\mathbf{c}, \mathtt{N}) \in \overline{R}$
- $(\mathrm{op}\{\mathbf{c}_1, \ldots, \mathbf{c}_{\mathrm{ar(op)}}\}, \mathtt{M}) \in \overline{R}$ if $\mathtt{M} \overset{\mathrm{op}_i}{\to} \mathtt{N}_i$ and $(\mathbf{c}_i, \mathtt{N}_i) \in \overline{R}$ for each $i = 1, \ldots, \mathrm{ar}(\mathtt{op})$
- $(\mathrm{op}\{\}, \mathtt{M}) \in \overline{R}$ if $\mathtt{M} \downarrow_{\mathrm{op}}$
- $(J(\bot), \mathtt{E}[\Omega_\tau]) \in \overline{R}$ for any type τ .

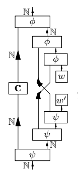
Note that a binary relation $\overline{R}$ is closed under behavioral equivalence. The operator $\bullet'$ in the above is *linear binary application* on T-transducers defined by

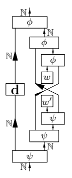
$$(X, c, x) \bullet' (Y, d, y) := \mathrm{Tr}^C_{A,B}((J(\mathrm{id}^*_B) \boxplus (Y, d, y)) \circ (X, c, x))$$

for two T-transducers $(X, c, x) \colon A{+}C \rightarrow\!\!\!\!\triangleright\, B{+}C$ and $(Y, d, y) \colon C \rightarrow\!\!\!\!\triangleright\, C$. It is a "linear" version of binary application $\bullet$ (from Section 4.2) where the countable copy operator F is excluded.

For each type τ a binary relation $R_\tau \subseteq \mathbf{Trans}(T)(\mathbb{N}, \mathbb{N}) \times \mathbf{Val}_\tau$ is inductively defined by

$$\begin{aligned}
R_{\mathtt{unit}} &= \{(J(\mathrm{id}^*_{\mathbb{N}}), \underline{*})\} \\
R_{\mathtt{nat}} &= \{(J(\mathrm{k}^*_n), \underline{n}) \mid n \in \mathbb{N}\} \\
R_{\sigma \to \tau} &= \{(J(\psi^*) \circ \mathbf{r} \circ J(\phi^*), \mathtt{V}) \\
&\qquad \mid \forall (\mathbf{d}, \mathtt{U}) \in R_\sigma.\, (\mathbf{r} \bullet \mathbf{d}, \mathtt{V}\,\mathtt{U}) \in \overline{R_\tau}\} \\
R_{\tau \times \sigma} &= \{(J(\phi^*) \circ (\mathbf{c} \boxplus \mathbf{d}) \circ J(\psi^*), \langle \mathtt{V}, \mathtt{U} \rangle) \\
&\qquad \mid (\mathbf{c}, \mathtt{V}) \in R_\tau, (\mathbf{d}, \mathtt{U}) \in R_\sigma\}
\end{aligned}$$

$R_{\tau + \sigma} = \{($  $, \mathtt{inl}_{\tau,\sigma}(\mathtt{M})) \mid (\mathbf{c}, \mathtt{M}) \in R_\tau\}$

$\cup \{($  $, \mathtt{inr}_{\tau,\sigma}(\mathtt{N})) \mid (\mathbf{d}, \mathtt{N}) \in R_\sigma\}$.

For a type environment $\Gamma = \mathtt{x_1} : \tau_1, \ldots, \mathtt{x_m} : \tau_m$, a binary relation R_Γ is defined by $R_{\tau_1} \times \cdots \times R_{\tau_m}$.

For a type judgment $\Gamma \vdash \mathtt{M} : \tau$ and T-transducers $\mathbf{c}_1, \ldots, \mathbf{c}_m : \mathbb{N} \rightarrow\!\!\!\!\triangleright\, \mathbb{N}$, let $(\!|\mathtt{M}|\!)[\vec{\mathbf{c}}] : \mathbb{N} \rightarrow\!\!\!\!\triangleright\, \mathbb{N}$ be a T-transducer defined by

$$(\!|\mathtt{M}|\!)[\vec{\mathbf{c}}] := (\cdots(((\!|\mathtt{M}|\!) \bullet' \mathbf{c}_1) \bullet' \mathbf{c}_2) \cdots \bullet' \mathbf{c}_m) \ .$$

We prove by induction on $\mathtt{M}$ that it satisfies

$$((\!|\mathtt{M}|\!)[\vec{\mathbf{c}}], \mathtt{M}[\vec{\mathtt{V}}/\Gamma]) \in \overline{R_\tau} \tag{3}$$

for any $(\vec{\mathbf{c}}, \vec{\mathtt{V}}) \in \overline{R_\Gamma}$. We note that if $(x, \mathtt{M}) \in \overline{R_{\mathtt{nat}}}$, then $[\![|\mathtt{M}|]\!] = x^\dagger$. In fact, by the definition of $\overline{R_{\mathtt{nat}}}$, it is easy to see that the relation $\overline{R_{\mathtt{nat}}}$ is a subset of $\{(x, \mathtt{M}) \mid [\![|\mathtt{M}|]\!] = x^\dagger\}$. The statement of Lemma 6.5 is a consequence of (3). In the proof of (3), properties of the component calculus summarized in Section 4.3 are exploited. Here, we only give outline of a proof for the case when $\mathtt{M}$ is equal to $\mathtt{rec}^{(i+1)}(\mathtt{f} : \tau \to \sigma, \mathtt{x} : \tau).\, \mathtt{N}$. We define a transducer $\mathbf{d}_i$ to be

$$\mathrm{Fix}^{(i)}_G((J(\phi^*) \boxplus J(u^*)) \circ (\!|\mathtt{N}[\vec{\mathbf{c}}]|\!) \circ (J(\psi^*) \boxplus J(v^*))).$$

We first prove

$$\begin{aligned}
&((J(\psi^*) \circ \mathbf{d}_{i+1} \circ J(\phi^*)) \bullet \mathbf{c}', \\
&\quad (\lambda \mathtt{x} : \tau.\, \mathtt{N}[\mathtt{rec}^{(i)}(\mathtt{f} : \tau \to \sigma, \mathtt{x} : \tau).\, \mathtt{N}/\mathtt{f}])[\vec{\mathtt{V}}/\Gamma]\,\mathtt{U}) \in \overline{R_\sigma}
\end{aligned} \tag{4}$$

for any $(\mathbf{c}', \mathtt{U}) \in R_\tau$. Since $\overline{R_{\tau \to \sigma}}$ is closed under behavioral equivalence and reductions, it is enough to check

$$\begin{aligned}
&((\!|\mathtt{N}|\!)[\vec{\mathbf{c}}, \mathbf{d}_i, \mathbf{c}'], \\
&\quad (\mathtt{N}[\mathtt{rec}^{(i)}(\mathtt{f} : \tau \to \sigma, \mathtt{x} : \tau).\, \mathtt{N}/\mathtt{f}])[\vec{\mathtt{V}}, \mathtt{U}/\Gamma, \mathtt{x}]) \in \overline{R_\tau}
\end{aligned}$$

for any $(\mathbf{c}', \mathtt{U}) \in R_\tau$. We can prove this by induction on i. Now, by the definition of $\overline{R_{\tau \to \sigma}}$ and (4), it follows that

$$\begin{aligned}
&(J(\psi^*) \circ \mathbf{d}_{i+1} \circ J(\phi^*), \\
&\quad (\lambda \mathtt{x} : \tau.\, \mathtt{N}[\mathtt{rec}^{(i)}(\mathtt{f} : \tau \to \sigma, \mathtt{x} : \tau).\, \mathtt{N}/\mathtt{f}])[\vec{\mathtt{V}}/\Gamma])
\end{aligned}$$

is in $R_{\tau \to \sigma}$, and therefore,

$$\begin{aligned}
&(J(\mathrm{h}^*) \bullet' (J(\psi^*) \circ \mathbf{d}_{i+1} \circ J(\phi^*)), \\
&\quad (\mathtt{rec}^{(i+1)}(\mathtt{f} : \tau \to \sigma, \mathtt{x} : \tau).\, \mathtt{N})[\vec{\mathtt{V}}/\Gamma]) \in \overline{R_{\tau \to \sigma}}.
\end{aligned}$$

Since the left component is behaviorally equivalent to $(\!|\mathtt{M}|\!)[\vec{\mathbf{c}}]$,

$$\begin{aligned}
&((\!|\mathtt{rec}^{(i+1)}(\mathtt{f} : \tau \to \sigma, \mathtt{x} : \tau).\, \mathtt{M}|\!)[\vec{\mathbf{c}}], \\
&\quad (\mathtt{rec}^{(i+1)}(\mathtt{f} : \tau \to \sigma, \mathtt{x} : \tau).\, \mathtt{N})[\vec{\mathtt{V}}/\Gamma])
\end{aligned}$$

is in $\overline{R_{\tau \to \sigma}}$. ☐

7. Executing Transducers

In this section we illustrate execution of transducers using the following example term

$$\begin{aligned}
&\mathtt{P} \equiv (\mathtt{rec}(\mathtt{flipLoop}, \mathtt{x}).\ \mathtt{Q})\ \underline{0}\ :\ \mathtt{nat} \\
&\text{where}\quad \mathtt{Q} \equiv \mathtt{x} \sqcup_{0.4} (\mathtt{flipLoop}\ (\mathtt{x} + \underline{1})).
\end{aligned}$$

The term $\mathtt{P}$ is a closed term of the language $\mathcal{L}_{\Sigma_{\mathrm{prob}}}$, where Σ_{prob} is the algebraic signature for probabilistic choice and supported by the subdistribution monad $\mathcal{D}$ (see Example 6.2). This term $\mathtt{P}$ intuitively flips an unfair coin repeatedly until we observe head, counting how many tails we observe. Indeed the effect value $|\mathtt{P}|$

can be seen as the infinite binary tree shown in Figure 9 and its

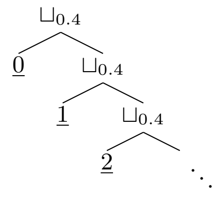


Figure 9: The Effect Value $|\mathtt{P}|$

interpretation $[\![|\mathtt{P}|]\!] \in \mathcal{D}\mathbb{N}$ is equal to the distribution over $\mathbb{N}$ such that $[\![|\mathtt{P}|]\!](\mathrm{enc}(\underline{n})) = 0.4 \times 0.6^n$ for each $n \in \mathbb{N}$.

As a consequence of Theorem 6.4 we can say that the $\mathcal{D}$-transducer $(\!|\mathtt{P}|\!) : \mathbb{N} \multimap \mathbb{N}$, depicted in Figure 10, outputs $\mathrm{enc}(\underline{n})$ with probability 0.4×0.6^n for each $n \in \mathbb{N}$. The execution of $(\!|\mathtt{P}|\!)$

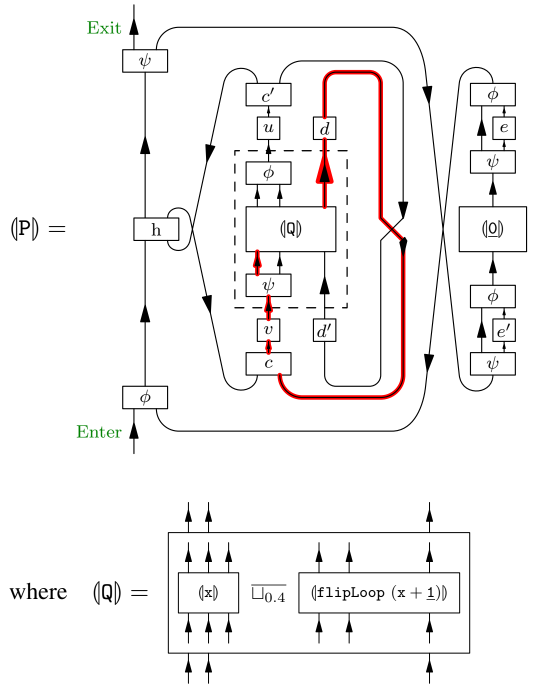


Figure 10: The $\mathcal{D}$-transducer $(\!|\mathtt{P}|\!) : \mathbb{N} \multimap \mathbb{N}$

can be visualized using a token that moves around the depiction along edges, updating the data (a natural number) carried by the token: it enters from the edge labeled with "Enter" carring the data $\mathrm{enc}(\underline{m})$, where m is a fixed natural number, and exits along the edge labeled with "Exit" with data $\mathrm{enc}(\underline{n})$ and probability 0.4×0.6^n for each $n \in \mathbb{N}$.

The way in which the token travels around captures dynamics of the evaluation of the term $\mathtt{P}$ in some sense. For example we can observe that the token enters a copy of the "subtransducer" $(\!|\mathtt{Q}|\!)$ as many times as the coin is flipped (i.e. the recursive function $\mathtt{rec}(\mathtt{flipLoop}, \mathtt{x}).\ \mathtt{Q}$ is called). Note that the transducer $(\!|\mathtt{Q}|\!)$ is within the dashed box in Figure 10 and hence "copied" by the operator F (see Section 4.2). Each copy has an index given by a natural number and, in fact, the token enters a different copy of $(\!|\mathtt{Q}|\!)$ for each time: namely it inters the g0-th copy for the first time, the $\mathtt{d}\langle \mathrm{g0}, \mathrm{g0}\rangle$-th copy for the second, the $\mathtt{d}\langle \mathtt{d}\langle \mathrm{g0}, \mathrm{g0}\rangle, \mathrm{g0}\rangle$-th copy for the third and so on. In this course, the token goes along the path indicated by the bold red arrows in Figure 10 (colors are available in an electronic edition), and enters a copy of $(\!|\mathtt{Q}|\!)$ as many times as the subterm `flipLoop` is recursively called (i.e. the coin results in tail).

Each copy of the transducer $(\!|\mathtt{Q}|\!)$ has the state space that is isomorphic to the set $\{*, \mathsf{L}, \mathsf{R}\}$; the transition diagram is given by

$$\begin{tikzcd} \mathsf{L} \arrow[loop left, "1"] & * \arrow[l, "0.4"'] \arrow[r, "0.6"] & \mathsf{R} \arrow[loop right, "1"] \end{tikzcd}$$

where labels denote probabilities. The copies of $(\!|\mathtt{Q}|\!)$ indeed record the history of probabilistic choices using their internal states: for example if the coin results in tail twice and in head at the third time (i.e. the term $\mathtt{P}$ is evaluated to $\underline{2}$), internal states of the copies result in as below.

index	internal state
g0	R
$\mathtt{d}\langle \mathrm{g0}, \mathrm{g0}\rangle$	R
$\mathtt{d}\langle \mathtt{d}\langle \mathrm{g0}, \mathrm{g0}\rangle, \mathrm{g0}\rangle$	L
others	$*$

Readers can see how exactly the token moves around using our tool *TtT*[3]—short for "Terms to Transducers." The tool automatically translates a given term of the language $\mathcal{L}_{\Sigma_{\mathrm{prob}}}$ and visualize execution of the resulting transducer by showing how a token moves around and updates its data.

8. Concluding Remark

In this paper we established a framework that gives GoI semantics for effectful programming language. Our framework accommodates various algebraic effects in a uniform way by employing categorical formalization of GoI and algebraic effects. This generality is inherited from the framework in our previous work; what is new in our current work is to accommodate recursion. Notably we gave two ways to interpret recursion—by Girard style and by Mackie style—and they were proved to be coincide. Additionally adequacy of our interpretation was proved. Our framework inherits another feature from the mGoI framework: that is, GoI interpretation of programs is given as transducers. In particular we gave the translation from effectful terms to transducers by means of a component calculus, even for recursion, with our prospect of developing a compilation technique for effectful programs and implementing them directly to hardware.

Let us give a remark on two styles of fixed point operator on transducers. In extracting a compilation technique from our framework and implementing it, one difficulty would arise from the use of countable copy operator F and countable parallel composition $\boxplus_{n\in\mathbb{N}}$ of the component calculus. The reason is that they both can inflate state spaces of transducers to infinite ones. From this point of view we would like to say that the Mackie style fixed point operator is a little more acceptable than the Girard style one, because the former includes only one occurrence of F while the latter includes one occurrence of $\boxplus_{n\in\mathbb{N}}$ and many occurrences of F. Therefore, although the Girard style fixed point operator exhibits convenient domain-theoretic properties, we would prefer the Mackie style fixed point operator in implementing our framework. That is why we used the Mackie style fixed point operator in the translation (Figure 7) of recursion.

Acknowledgments

The authors are supported by the JSPS-INRIA Bilateral Joint Research Project "CRECOGI," and by Grant-in-Aid No. 15K11984, JSPS. K.M. and I.H. are supported by Grants-in-Aid No. 24680001 & 15KT0012, JSPS; N.H. is supported by Grant-in-Aid No. 26730004, JSPS.

[3] http://koko-m.github.io/TtT/

References

[1] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000.

[2] S. Abramsky, E. Haghverdi, and P. J. Scott. Geometry of Interaction and linear combinatory algebras. *Math. Struct. in Comp. Sci.*, 12(5): 625–665, 2002.

[3] U. Dal Lago and U. Schöpp. Type inference for sublinear space functional programming. In *Programming Languages and Systems*, volume 6461 of *Lecture Notes in Computer Science*, pages 376–391. Springer Berlin Heidelberg, 2010.

[4] V. Danos and L. Regnier. Local and asynchronous beta-reduction (an analysis of girard's execution formula). In *Logic in Computer Science, 1993. LICS '93., Proceedings of Eighth Annual IEEE Symposium on*, pages 296–306, 1993.

[5] J. Egger, R. E. Møgelberg, and A. Simpson. Enriching an effect calculus with linear types. In E. Grädel and R. Kahle, editors, *Computer Science Logic*, volume 5771 of *Lecture Notes in Computer Science*, pages 240–254. Springer Berlin Heidelberg, 2009.

[6] M. Fernández and I. Mackie. Call-by-value λ-graph rewriting without rewriting. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Graph Transformation*, volume 2505 of *Lecture Notes in Computer Science*, pages 75–89. Springer Berlin Heidelberg, 2002.

[7] D. Ghica. Geometry of Synthesis: a structured approach to VLSI design. In *POPL 2007*, pages 363–375. ACM, 2007.

[8] D. R. Ghica and A. I. Smith. Geometry of synthesis II: from games to delay-insensitive circuits. *Electr. Notes Theor. Comput. Sci.*, 265: 301–324, 2010.

[9] D. R. Ghica and A. I. Smith. Geometry of synthesis III: resource management through type inference. In T. Ball and M. Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 345–356. ACM, 2011.

[10] D. R. Ghica, A. I. Smith, and S. Singh. Geometry of Synthesis IV: compiling affine recursion into static hardware. In *ICFP*, pages 221–233, 2011.

[11] J.-Y. Girard. Geometry of interaction 1: Interpretation of system F. In S. V. R. Ferro, C. Bonotto and A. Zanardo, editors, *Logic Colloquium '88 Proceedings of the Colloquium held in Padova*, volume 127 of *Studies in Logic and the Foundations of Mathematics*, pages 221–260. Elsevier, 1989.

[12] J. Y. Girard. Geometry of interaction 2: Deadlock-free algorithms. In *Proceedings of the International Conference on Computer Logic*, COLOG-88, pages 76–93, New York, NY, USA, 1990. Springer-Verlag New York, Inc. ISBN 0-387-52335-9.

[13] E. Haghverdi. Towards a geometry of recursion. *Theor. Comput. Sci.*, 412(20):2015–2028, Apr. 2011. ISSN 0304-3975. .

[14] M. Hasegawa. On traced monoidal closed categories. *Math. Struct. in Comp. Sci.*, 19(2):217–244, 2009.

[15] N. Hoshino. A modified GoI interpretation for a linear functional programming language and its adequacy. In *FoSSaCS 2011*, volume 6604 of *Lect. Notes Comp. Sci.*, pages 320–334. Springer, 2011.

[16] N. Hoshino, K. Muroya, and I. Hasuo. Memoryful Geometry of Interaction: from coalgebraic components to algebraic effects. In *CSL-LICS 2014*, page 52. ACM, 2014.

[17] J. Hyland and C.-H. Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, 2000.

[18] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Math. Proc. Cambridge Phil. Soc.*, 119(3):447–468, 1996.

[19] U. D. Lago, C. Faggian, B. Valiron, and A. Yoshimizu. Parallelism and synchronization in an infinitary context. In *LICS*, 2015.

[20] O. Laurent. A token machine for full Geometry of Interaction. In *TLCA*, pages 283–297, 2001.

[21] S. Mac Lane. *Categories for the working mathematician*. Springer, 1998.

[22] I. Mackie. The Geometry of Interaction machine. In *POPL 1995*, pages 198–208. ACM, 1995.

[23] E. Moggi. Computational lambda-calculus and monads. *Tech. Report*, pages 1–23, 1988.

[24] J. S. Pinto. *Implantation Parallèle avec la Logique Linéaire (Applications des Réseaux d'Interaction et de la Géométrie de l'Interaction)*. PhD thesis, École Polytechnique, 2001. Main text in English.

[25] G. Plotkin and J. Power. Adequacy for algebraic effects. In *FoSSaCS 2001*, volume 2030 of *Lect. Notes Comp. Sci.*, pages 1–24. Springer, 2001.

[26] G. Plotkin and J. Power. Algebraic operations and generic effects. *Appl. Categorical Struct.*, 11(1):69–94, 2003.

[27] J. Power and E. Robinson. Premonoidal categories and notions of computation. *Math. Struct. in Comp. Sci.*, 7(5):453–468, 1997.

[28] U. Schöpp. Computation-by-interaction with effects. In H. Yang, editor, *Programming Languages and Systems*, volume 7078 of *Lecture Notes in Computer Science*, pages 305–321. Springer Berlin Heidelberg, 2011.

[29] U. Schöpp. Call-by-value in a basic logic for interaction. In J. Garrigue, editor, *Programming Languages and Systems*, volume 8858 of *Lecture Notes in Computer Science*, pages 428–448. Springer International Publishing, 2014.

[30] U. Schöpp. From call-by-value to interaction by typed closure conversion. In *APLAS*, 2015. To appear.