# Recursion and Adequacy in Memoryful Geometry of Interaction
# 記憶つき相互作用の幾何における再帰と妥当性

by

Koko Muroya
室屋晃子

A Master Thesis
修士論文

Submitted to
the Graduate School of the University of Tokyo
on February 26, 2016
in Partial Fulfillment of the Requirements
for the Degree of Master of Information Science and Technology
in Computer Science

Thesis Supervisor: Ichiro Hasuo　蓮尾一郎
Associate Professor of Computer Science

## ABSTRACT

The *Geometry of Interaction (GoI)* framework, introduced by Girard, provides semantics of linear logic proofs originally, and of functional programs as well via the Curry-Howard correspondence. Notably the obtained program semantics—what we shall call “GoI semantics”—captures dynamics of program execution while the semantics is denotational and compositional. This feature of GoI semantics leads to its executable representations and hence its practical applications, e.g. Mackie’s compilation technique and Ghica’s high-level synthesis technique.

One of theoretical challenges in GoI semantics is accommodation of computational effects such as (probabilistic) nondeterminism, exception and local/global states. For this challenge the *memoryful Geometry of Interaction (mGoI)* framework was developed by Hoshino, Muroya and Hasuo. It accommodates a class of computational effects, namely algebraic effects studied by Plotkin and Power, that includes (probabilistic) nondeterminism, exception and global states. The mGoI framework provides the GoI semantics represented by *transducers* that are “effectful” extensions of stream transducers or Mealy machines.

The current work is a theoretical extension of the mGoI framework to accommodate recursion, that is lacking in the original framework. We first extend the GoI semantics, provided by the original mGoI framework, in two styles that we call the Girard and Mackie styles. We then show the coincidence of these two styles and prove adequacy of the extended GoI semantics with respect to Plotkin and Power’s operational semantics.



## 論文要旨

関数型プログラムに対する意味論のひとつを与える**相互作用の幾何 (GoI)** は、線形論理の証明に対する意味論を与える Girard による枠組みを Curry-Howard 対応を用いて応用したものである。この GoI が与えるプログラム意味論 (ここでは GoI 意味論と呼ぶことにする) には、表示的・要素還元的な意味論でありながらプログラム実行の動的な性質を捉えているという特徴がある。この特徴を活かし GoI 意味論に実行可能な表現を与えることで、Mackie のコンパイル技術や Ghica の高位合成技術といった実用的な応用が生まれている。

GoI 意味論において理論的な難しさを持つもののひとつに、(確率的) 非決定性、例外、局所/大域変数といった計算副作用がある。この計算副作用に対処するために星野、室屋、蓮尾が考案したのが**記憶付き相互作用の幾何 (mGoI)** である。mGoI では Plotkin と Power によって提唱された、(確率的) 非決定性、例外、局所変数などからなる代数的副作用と呼ばれる計算副作用のクラスを扱うことができる。また mGoI が与える GoI 意味論は、ミーリーマシンの計算副作用への拡張ともいえる**トランスデューサ**によって表現される。

これまで mGoI では扱えなかった再帰を扱うために、この論文では mGoI の理論的な拡張を行う。mGoI の与える GoI 意味論に対して、まず Girard 様式・Mackie 様式という 2 通りの拡張を与える。それら 2 様式の一致を示したのち、得られた GoI 意味論の妥当性を Plotkin と Power の操作的意味論に対して証明する。

# Acknowledgements

My thanks are due to Naohiko Hoshino for helpful comments and discussions. It was always challenging but fruitful for me to jointly work with him. I am also indebted to Dan Ghica for useful discussions that gave me many new insights. My deepest gratitude goes to my supervisor Ichiro Hasuo for everything, from his insightful comments and stimulating advice to his demonstration of how it is being a researcher. This work would not have been completed without daily chats and discussions with the past and present members of Hasuo Laboratory. I finally would like to thank my family, friends and all those who encouraged and supported me.

# Contents


**1 Introduction** **1**
1.1 Geometry of Interaction . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2 Memoryful Geometry of Interaction . . . . . . . . . . . . . . . . . . 2
1.3 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

**2 Terms with Algebraic Effects** **4**
2.1 Syntax and Operational Semantics of Target Language . . . . . . . 4
2.1.1 Target Language $\mathcal{L}_\Sigma$ . . . . . . . . . . . . . . . . . . . . . . . 4
2.1.2 Operational Semantics . . . . . . . . . . . . . . . . . . . . . . 5
2.2 Supported Algebraic Signatures . . . . . . . . . . . . . . . . . . . . 7
2.2.1 Monads on the Category **Set** . . . . . . . . . . . . . . . . . . 8
2.2.2 Algebraic Operations on Monads . . . . . . . . . . . . . . . . 10
2.2.3 Algebraic Signatures Supported by Monads . . . . . . . . . . 11

**3 Component Calculus over Transducers** **12**
3.1 Categorical Geometry of Interaction . . . . . . . . . . . . . . . . . . 12
3.2 Component Calculus in the mGoI Framework . . . . . . . . . . . . 13
3.2.1 Operators on Transducers . . . . . . . . . . . . . . . . . . . . 14
3.2.2 Primitive "Memoryless" Transducers . . . . . . . . . . . . . . 17
3.2.3 "Category" of Transducers . . . . . . . . . . . . . . . . . . . 19
3.3 Extension of Component Calculus . . . . . . . . . . . . . . . . . . . 20
3.3.1 Countable Parallel Composition $\boxplus_{n\in\mathbb{N}}$ . . . . . . . . . . . . 20
3.3.2 Girard Style Fixed Point Operator $\mathrm{Fix}_G$ . . . . . . . . . . . . 21
3.3.3 Induced $\omega$-cpo Structure on Transducers . . . . . . . . . . . 22
3.3.4 Mackie Style Fixed Point Operator $\mathrm{Fix}_M$ . . . . . . . . . . 25

**4 Adequate Translation of Effectful Terms to Transducers** **29**
4.1 Translation to Transducers . . . . . . . . . . . . . . . . . . . . . . . 29
4.1.1 Underlying Categorical Model . . . . . . . . . . . . . . . . . . 33
4.2 Adequacy of Translation . . . . . . . . . . . . . . . . . . . . . . . . . 34
4.2.1 Proof of Adequacy . . . . . . . . . . . . . . . . . . . . . . . . 35
4.3 Execution of Resulting Transducers . . . . . . . . . . . . . . . . . . 38
4.3.1 Execution Cost . . . . . . . . . . . . . . . . . . . . . . . . . . 40

**5 Conclusion and Future Work** **42**

**References** **44**

# Chapter 1

# Introduction

## 1.1 Geometry of Interaction

Girard's *Geometry of Interaction (GoI)* framework [11] originally provides semantics of linear logic proofs, and can also provide semantics of functional programs via the Curry-Howard correspondence of proofs and programs. Given a program, the GoI framework calculates its "execution path" that is invariant under $\beta$-reductions. They are precisely given as elements of a $C^*$-algebra (or a dynamic algebra), and can be understood as "valid paths" on the type derivation trees of programs or as sequences of interactions between components of programs.

Several program semantics are proposed by exploiting the evaluation paths. One is *graph rewriting semantics* [13], in which programs are translated to graphs that intuitively represents the structure of type derivation trees, and reduction of graphs respects execution paths. Another is *token machine semantics* [24, 4], in which programs are translated to abstract machines—called token machines—that directly calculates execution paths. A token machine can be expressed by a graph, and its execution can be depicted using a token moving around the graph and updating its data. These program semantics capture dynamics of program evaluation in some sense. In particular function application is interpreted as interactions of a function and its arguments in token machine semantics.

Since token machine semantics gives "executable" interpretation of programs, it is exploited to obtain compilation techniques of functional programs. For example Mackie gives in [24] a compilation technique of PCF by implementing token machines in assembly language, and Ghica et al. in their series of work [6, 7, 8, 10] gives a hardware synthesis technique of functional programs by directly implementing token machines on hardware. Compilation techniques extracted from token machine semantics can be defined compositionally and their correctness is guaranteed by their definition, because token machine semantics is denotational.

Token machine semantics has not only practical applications but also the categorical formalization. Abramsky, Haghverdi and Scott categorically formalizes GoI, in particular token machine semantics, in [1]. They namely introduces the notion of *GoI situation* as an categorical axiomatization of token machines together with constructions and axioms of them, and shows how a GoI situation yields a model of untyped $\lambda$-calculus. This categorical formalization of GoI, that we shall call *categorical GoI*, enables us to give variations of token machine semantics, in which the notion of token machine is variously extended, in a uniform way. For example Hasuo and Hoshino proposes token machine semantics of quantum computation in [16], with equipping token machines with "quantum branching."

### 1.2 Memoryful Geometry of Interaction

Computational effects are characteristics of computer programs such that (probabilistic) nondeterminism, exception, local/global states, I/O and so on. While they are well accommodated and utilized in practical programming languages, it is well known that we need some additional mechanism to give their denotational semantics. There have been proposed several categorical approaches to model computational effects, in particular the monadic approach by Moggi [25] and the algebraic approach by Plotkin and Power [30]. These categorical approaches enjoy genericity, that is, they enable us to model various computational effects in a uniform way.

The *memoryful Geometry of Interaction (mGoI)* framework, developed by Hoshino, Muroya and Hasuo in [18], provides token machine semantics of effectful computations. It combines categorical GoI with Plotkin and Power's algebraic approach to categorically model computational effects, and therefore can accommodate *algebraic effects* in a uniform way. Algebraic effects are computational effects that can be modeled by Plotkin and Power's approach, and they are generated only by operations specified by an algebraic signature. They include for example (probabilistic) nondeterminism, exception, global states and I/O.

Token machines in the mGoI framework are not only "effectful" but also "memoryful." Hasuo and Hoshino suggest in [16] that "effectful" token machines can be used to give token machine semantics of effectful computations, however they observe difficulty in controlling generation of effects. The mGoI framework equips "effectful" token machines with internal states (or "memories") so that they can control generation of effects. The resulting token machines are precisely called *transducers*.

We explain how transducers utilize their internal states using an example. Let

$$\mathtt{P} \equiv (\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x})\ \mathtt{choose}(\underline{0}, \underline{1})$$

be a $\lambda$-term with the nondeterministic choice operation `choose`. It is translated in the mGoI framework to a transducer $(\!|\mathtt{P}|\!)$ that behaves nondeterministically. Execution of the transducer can be essentially understood as the following sequence of interactions between two transducers $(\!|\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x}|\!)$ and $(\!|\mathtt{choose}(\underline{0}, \underline{1})|\!)$. The former transducer is "memoryless," and the latter has the state space $\{*, \mathsf{L}, \mathsf{R}\}$ with the initial state $*$.

1. $(\!|\lambda \mathtt{x}.\, \mathtt{x}+\mathtt{x}|\!)$ requires output of $(\!|\mathtt{choose}(\underline{0}, \underline{1})|\!)$ as the value of the left argument $\mathtt{x}$.

2. $(\!|\mathtt{choose}(\underline{0}, \underline{1})|\!)$ makes a nondeterministic choice. According to the choice, it changes its internal state from $*$ to either $\mathsf{L}$ or $\mathsf{R}$, and outputs either 0 or 1.

3. $(\!|\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x}|\!)$ requires output of $(\!|\mathtt{choose}(\underline{0}, \underline{1})|\!)$ as the value of the right argument $\mathtt{x}$.

4. $(\!|\mathtt{choose}(\underline{0}, \underline{1})|\!)$ consults its internal states, and outputs 0 if the state is $\mathsf{L}$ and 1 if the state is $\mathsf{R}$, without making any nondeterministic choice.

5. $(\!|\lambda \mathtt{x}.\, \mathtt{x}+\mathtt{x}|\!)$ outputs the sum of the first and second output of $(\!|\mathtt{choose}(\underline{0}, \underline{1})|\!)$.

In this way the transducer $(\!|\mathtt{P}|\!)$ outputs either 0 or 1, that corresponds to the result of call-by-value evaluation of the term $\mathtt{P}$. Because of the "call-by-name" nature of GoI, the transducer $(\!|\lambda \mathtt{x}.\, \mathtt{x} + \mathtt{x}|\!)$ requires output of the transducer $(\!|\mathtt{choose}(\underline{0}, \underline{1})|\!)$

as many times as the variable $\mathtt{x}$ occurs in the subterm $\mathtt{x} + \mathtt{x}$. Therefore if we adopt the call-by-value evaluation strategy, we need to prevent the transducer $(\!|\mathtt{choose}(\underline{0}, \underline{1})|\!)$ from making nondeterministic choices every time its output is required. The transducer utilizes its internal state to memorize the result of its choice and avoid making another choice.

Note that we need to control effectful behavior of transducers even if we do not adopt the call-by-value evaluation strategy. For example in the translation of the $\lambda$-term

$$\mathtt{choose}(\lambda\mathtt{x}.\,\mathtt{x},\ \lambda\mathtt{x}.\,\mathtt{x} + \mathtt{x})\ \underline{1}\ ,$$

we need to make sure the transducer $(\!|\mathtt{choose}(\lambda\mathtt{x}.\,\mathtt{x},\ \lambda\mathtt{x}.\,\mathtt{x} + \mathtt{x})|\!)$ behaves consistently as either $(\!|\lambda\mathtt{x}.\,\mathtt{x}|\!)$ or $(\!|\lambda\mathtt{x}.\,\mathtt{x} + \mathtt{x}|\!)$. The consistent behavior is ensured in the mGoI framework by utilizing internal states.

Our final remark on the mGoI framework is that it exploits a *coalgebraic component calculus* based on [2, 17]. The mGoI framework provides token machine semantics of effectful computations, namely computations with algebraic effects, in which effectful $\lambda$-terms are translated to transducers. The translation is defined by means of a coalgebraic component calculus that gives a set of operators on coalgebraic components, namely transducers.

### 1.3 Contributions

The current work extends the mGoI framework to accommodate *recursion* that is practically important in functional programming but lacking in the mGoI framework. Our framework provides token machine semantics of effectful computations with recursion, inheriting the features of the mGoI framework reviewed in the last section. Namely in our framework, algebraic effects are accommodated in a uniform way by exploiting category theory, and effectful $\lambda$-terms, possibly with recursion, are translated to transducers by means of a coalgebraic component calculus.

To translate recursion by means of a coalgebraic component calculus, we extend the component calculus developed in the mGoI framework by introducing two styles of "fixed point" operators on transducers. They are namely the *Girard style* and *Mackie style* fixed point operators, and defined by following existing approaches to accommodate recursion in GoI. The Girard style fixed point operator enables us to interpret recursion as a limit of finite approximations, as much like Girard's domain-theoretic approach in [12]. The Mackie style fixed point operator enables us to translate recursion by adding "self-loops" to transducers, following Mackie's approach in [24] to give token machine semantics for recursion in the implementable way.

One of our main contributions is to show the coincidence of these two styles of fixed point operators. Thanks to this coincidence result we can enjoy useful features of both two styles in translating recursion. The domain-theoretic properties of the Girard style fixed point operator are exploited to obtain the adequacy result, while the Mackie style fixed point operator gives simpler, and more easily implementable, translation of recursion.

The other of our main contributions is to prove adequacy of our translation of effectful $\lambda$-terms to transducers. Our adequacy result is with respect to Plotkin and Power's operational semantics given in [29].

The current work is based on the joint work with Naohiko Hoshino and Ichiro Hasuo in [26].

# Chapter 2

# Terms with Algebraic Effects

## 2.1 Syntax and Operational Semantics of Target Language

In this section we give syntax and operational semantics of our target language $\mathcal{L}_\Sigma$, which are slight adaptations of those presented by Plotkin and Power in [29]. The language $\mathcal{L}_\Sigma$ is an extension of Moggi's (call-by-value) computational $\lambda$-calculus [25] by operations that generate computational effects called *algebraic effects*, by arithmetic primitives and additionally by recursion. Main differences between our language and the Plotkin and Power's one are generalization of the base type `bool` to coproduct types $\tau+\sigma$ and introduction of the binary summation $+$ as an arithmetic primitive.

### 2.1.1 Target Language $\mathcal{L}_\Sigma$

Our target language $\mathcal{L}_\Sigma$ is parameterized by an algebraic signature $\Sigma$ that consists of *operations* `op`, each of which is accompanied by its *arity* $\mathrm{ar}(\mathtt{op})$. All arities are restricted to be finite for simplicity, although infinite arities can be accommodated in our framework straightforwardly and their syntactical account is discussed in [30]. For an operation $\mathtt{op} \in \Sigma$ we often write $\mathtt{op}^+$ if its arity is positive and $\mathtt{op}^0$ if its arity is zero. An algebraic signature $\Sigma$ determines which effectful computation the language $\mathcal{L}_\Sigma$ is for, by specifying operations that generate effects.

We give examples of algebraic signatures below and discuss what signatures can be used in our framework later.

**Example 2.1.1.** Here are our leading examples of algebraic signatures taken from [30].

- The signature $\Sigma_{\text{except}} = \{\mathtt{raise}_e \mid e \in E\}$ is for *exceptions* where $E$ is a set that specifies possible exceptions. Each nullary operation $\mathtt{raise}_e$ raises an exception $e \in E$.

- The signature $\Sigma_{\text{nondet}} = \{\mathtt{choose}\}$ is for *nondeterminism.* The binary operation `choose` performs a nondeterministic choice, i.e. either its left argument or its right argument is nondeterministically chosen and evaluated.

- The signature $\Sigma_{\text{prob}} = \{\mathtt{choose}_p \mid p \in [0,1]\}$ is for *probabilistic choice.* For each $p \in [0,1]$, the binary operation $\mathtt{choose}_p$ performs a probabilistic choice in which its left argument is evaluated with probability $p$ and its right argument is with probability $1-p$.

- The signature $\Sigma_{\text{glstate}} = \{\mathtt{lookup}_\ell \mid \ell \in \mathit{Loc}\} \cup \{\mathtt{update}_{\ell,v} \mid \ell \in \mathit{Loc}, v \in \mathit{Val}\}$ is for *actions on global states*, where a set *Loc* specifies locations of global states and a finite set *Val* specifies stored values. Each $|\mathit{Val}|$-ary

operation $\texttt{lookup}_\ell$ evaluates one of its arguments according to a stored value of the global state of location $\ell \in \mathit{Loc}$. The unary operation $\texttt{update}_{\ell,v}$, for each $\ell \in \mathit{Loc}$ and $v \in \mathit{Val}$, first store the value $v$ to the global state of location $\ell$ and then evaluates its (unique) argument.

Types $\tau$ and terms $\texttt{M}$ of the language $\mathcal{L}_\Sigma$ are defined by the following BNF's:

$$\tau ::= \texttt{unit} \mid \texttt{nat} \mid \tau \to \tau \mid \tau \times \tau \mid \tau + \tau$$

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

where $\mathbf{Var}$ is a set of variables, $\mathbb{N}$ is the set of natural numbers and $\texttt{op} \in \Sigma$ is an operation. In addition to ordinal $\lambda$-calculus the language $\mathcal{L}_\Sigma$ accommodates recursion, algebraic effects via operations in $\Sigma$, pairs, pattern matching and arithmetic. As usual, substitution $\texttt{M}[\texttt{N}/\texttt{x}]$ is inductively defined and a term $\texttt{M}$ is called *closed* if it has no free variables.

Typing rules are defined as below, where $\Gamma$ denotes a finite list $\texttt{x}_1 : \tau_1, \ldots, \texttt{x}_m : \tau_m$ of some length $m \in \mathbb{N}$.

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

$$\frac{\Gamma, \texttt{f} : \sigma \to \tau, \texttt{x} : \sigma \vdash \texttt{M} : \tau}{\Gamma \vdash \texttt{rec}(\texttt{f} : \sigma \to \tau, \texttt{x} : \sigma).\,\texttt{M} : \sigma \to \tau} \quad \frac{\Gamma \vdash \texttt{M}_i : \tau \ \ (i = 1, \ldots, \mathrm{ar}(\texttt{op}))}{\Gamma \vdash \texttt{op}^{+}(\texttt{M}_1, \ldots, \texttt{M}_{\mathrm{ar}(\texttt{op})}) : \tau} \quad \frac{}{\Gamma \vdash \texttt{op}^{0}() : \tau}$$

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

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

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

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

All arguments of an operation $\texttt{op}^{+}$ are required to have the same type and a term $\texttt{op}^{0}()$ can be arbitrarily typed. All other rules are usual. A term $\texttt{M}$ is called *well-typed* if there exists a derivable type judgement $\Gamma \vdash \texttt{M} : \tau$ for some list $\Gamma$ and some type $\tau$, and we restrict all terms to be well-typed.

### 2.1.2 Operational Semantics

In [29] Plotkin and Power define two kinds of operational semantics of their language, namely the *small step* and the *medium step* operational semantics, and additionally the notion of *effect value* aiming at big step operational semantics. We adapt their definitions for our language $\mathcal{L}_\Sigma$.

We begin with defining values $\texttt{V}$, evaluation contexts $\texttt{E}$ and redexes $\texttt{R}$ by the following BNF's.

$$\texttt{V} ::= \texttt{x} \in \mathbf{Var} \mid \lambda\texttt{x} : \sigma.\,\texttt{M} \mid \underline{*} \mid \langle \texttt{V}, \texttt{V} \rangle \mid \texttt{inl}_{\tau,\sigma}(\texttt{V}) \mid \texttt{inr}_{\tau,\sigma}(\texttt{V}) \mid \underline{n} \in \mathbb{N}$$

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

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

Any closed term $\mathtt{M}$ can be uniquely decomposed using these notions if it is not a value. Precisely it satisfies just one of the following.

- $\mathtt{M} \equiv \mathtt{V}$ for a unique value $\mathtt{V}$
- $\mathtt{M} \equiv \mathtt{E}[\mathtt{R}]$ for a unique evaluation context $\mathtt{E}$ and a unique redex $\mathtt{R}$
- $\mathtt{M} \equiv \mathtt{E}[\mathtt{op}^0()]$ for a unique evaluation context $\mathtt{E}$ and a unique nullary operation $\mathtt{op}^0 \in \Sigma$

For closed redexes, two kinds of transition relations are defined as below.

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

The unlabeled "pure" transition relation $\to$ is for ordinal $\beta$-reduction; note that we adopt the call-by-value evaluation strategy. A family of labeled "effectful" transition relation $\{\overset{\mathtt{op}_i}{\to}\}_{i=1}^{\mathrm{ar}(\mathtt{op})}$ is for reduction according effects generated by the operator $\mathtt{op}^+$. Additionally for a term $\mathtt{op}^0()$, a labeled "effectful termination" predicate is defined by $\mathtt{op}^0() \downarrow_{\mathtt{op}}$.

These relations and predicates specify the two operational semantics via the unique decomposition of terms. For closed terms that are not values, the small step operational semantics are defined by lifting the transition relations and predicates as below.

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

For a closed term $\mathtt{M}$ its medium step operational semantics is defined by:

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

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

**Lemma 2.1.2.** (I) Any closed term $\mathtt{M}$ of type $\tau$ satisfies just one of the following.

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

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

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

Although we do not give big step operational semantics, we utilize the notion of effect value that aims at it. Effect values are defined to be elements of continuous $\Sigma$-algebras. A continuous $\Sigma$-algebra $\mathcal{A}$ is an $\omega$-cppo (i.e. $\omega$-cpo with the least element $\Omega$), with each operation $\mathtt{op} \in \Sigma$ identified with a continuous function from the $\mathrm{ar}(\mathtt{op})$-fold product of $\mathcal{A}$ to $\mathcal{A}$ itself. In particular any set $X$ induces the free continuous $\Sigma$-algebra $CT_\Sigma(X)$ over it, and so does the set $\mathbf{Val}_\tau$ of values of type $\tau$. For a closed term $\mathtt{M}$ of type $\tau$, its effect value $|\mathtt{M}| \in CT_\Sigma(\mathbf{Val}_\tau)$ is defined to be the limit of "finite approximations" $|\mathtt{M}|^{(k)} \in CT_\Sigma(\mathbf{Val}_\tau)$ as below.

$$
\begin{aligned}
|\mathtt{M}|^{(0)} &:= \Omega \\
|\mathtt{M}|^{(k+1)} &:= \begin{cases} \mathtt{V} & (\mathtt{M} \Rightarrow \mathtt{V}) \\ \mathtt{op}^+(|\mathtt{N}_1|^{(\mathtt{k})}, \dots, |\mathtt{N}_{\mathrm{ar}(\mathtt{op})}|^{(\mathtt{k})}) & (\mathtt{M} \overset{\mathtt{op}_i}{\Rightarrow} \mathtt{N}_i \text{ for each } i = 1, \dots, \mathrm{ar}(\mathtt{op})) \\ \mathtt{op}^0() & (\mathtt{M} \Downarrow_{\mathtt{op}}) \\ \Omega & (\mathtt{M} \Uparrow) \end{cases} \\
|\mathtt{M}| &:= \sup_{k \in \omega} |\mathtt{M}|^{(k)}
\end{aligned}
$$

Intuitively elements of the free continuous $\Sigma$-algebra $CT_\Sigma(\mathbf{Val}_\tau)$ can be understood as possibly infinite $\Sigma$-branching trees over the set $\mathbf{Val}_\tau \cup \{\Omega\}$. Therefore an effect value $|\mathtt{M}|$ can be understood as a $\Sigma$-branching tree that is equal to:

$$
\begin{cases} \mathtt{V} & \text{if } \mathtt{M} \Rightarrow \mathtt{V} \\ \mathtt{op}^+ \text{ with children } |\mathtt{N}_1| \quad \dots \quad |\mathtt{N}_{\mathrm{ar}(\mathtt{op})}| & \text{if } \mathtt{M} \overset{\mathtt{op}_i}{\Rightarrow} \mathtt{N}_i \text{ for each } i = 1, \dots, \mathrm{ar}(\mathtt{op}) \\ \mathtt{op}^0() & \text{if } \mathtt{M} \Downarrow_{\mathtt{op}} \\ \Omega & \text{if } \mathtt{M} \Uparrow \quad . \end{cases}
$$

## 2.2 Supported Algebraic Signatures

This section describes what algebraic signatures can be used in our framework: these signatures are characterized by means of *monads*. We begin with some facts about monads and their Kleisli categories that Moggi [25] utilizes to capture effectful computations, and provide some requirements for monads to develop our framework. After recalling how algebraic signatures are modeled using monads in Plotkin and Power's series of work [29, 30], we characterize our supported signatures by monads. This section utilizes some categorical stuff without explanations, and readers are referred to e.g. [23] for basic categorical notions and their precise definitions.

### 2.2.1 Monads on the Category Set

A *monad* $T$ on a category $\mathbb{C}$ is an endofunctor on $\mathbb{C}$, equipped with two natural transformations $\eta\colon 1_{\mathbb{C}} \Rightarrow T$ and $\mu\colon T^2 \Rightarrow T$ subject to certain coherence conditions. It induces the Kleisli category $\mathbb{C}_T$ whose objects are the same objects as $\mathbb{C}$ and whose arrows $X \to_T Y$ are $\mathbb{C}$-arrows in the form of $X \to TY$. Moggi's idea in [25] is to represent effectful computations as $\mathbb{C}_T$-arrows, in contrast to pure computations represented as $\mathbb{C}$-arrows. For example, nondeterministic computation can be represented as functions in the form of $X \to \mathcal{P}Y$, where $\mathcal{P}Y$ denotes the powerset of $Y$; given input in $X$, a function $X \to \mathcal{P}Y$ returns a subset $V \subseteq Y$ of possible output. Functions $X \to \mathcal{P}Y$ are precisely arrows of the Kleisli category $\mathbf{Set}_{\mathcal{P}}$, where $\mathcal{P}$ is the powerset monad $\mathcal{P}Y = \{V \subseteq Y\}$ and $\mathbf{Set}$ is the category of sets and functions.

The natural transformation $\mu\colon T^2 \Rightarrow T$, called *multiplication*, is used to "flatten" multiple effects that occur in composing two effectful computations. Namely, composition $g \circ_T f\colon X \to_T Z$ of two $\mathbb{C}_T$-arrows $f\colon X \to_T Y$ and $g\colon Y \to_T Z$ is defined by composing $\mathbb{C}$-arrows:

$$g \circ_T f\colon\ X \xrightarrow{f} TY \xrightarrow{Tg} T^2Z \xrightarrow{\mu_Z} TZ \quad .$$

Pure computations can be regarded as effectful computations that in fact generate no effects. The other natural transformation $\eta\colon 1_{\mathbb{C}} \Rightarrow T$, called *unit*, is used to "lift" pure computations represented as $\mathbb{C}$-arrows to effectful computations represented as $\mathbb{C}_T$-arrows. Each $\mathbb{C}$-arrow $f\colon X \to Y$ is lifted to a $\mathbb{C}_T$-arrow $f^*\colon X \to_T Y$ defined by composing $\mathbb{C}$-arrows:

$$f^*\colon\ X \xrightarrow{f} Y \xrightarrow{\eta_Y} TY \quad .$$

Now we focus on a monad $T$ on the particular category $\mathbf{Set}$ of sets and functions. The Kleisli category $\mathbf{Set}_T$ inherits both finite coproducts $(+, \emptyset)$ and countable ones $\coprod$ from $\mathbf{Set}$, with injections preserved by the lifting $(-)^*$ from $\mathbf{Set}$-arrows to $\mathbf{Set}_T$-arrows. Let $1$ be a singleton $\{*\}$. The category $\mathbf{Set}$ has finite products $(\times, 1)$ and any monad $T$ on it comes with tensorial strengths [20] $\mathrm{st}_{X,Y}\colon X \times TY \to T(X \times Y)$ and $\mathrm{st}'_{X,Y}\colon TX \times Y \to T(X \times Y)$. They induce the premonoidal structure [31] of the Kleisli category $\mathbf{Set}_T$: for any $A \in \mathbf{Set}$ and $\mathbf{Set}_T$-arrow $f\colon X \to_T Y$, two $\mathbf{Set}_T$-arrows $A \otimes f\colon A \times X \to_T A \times Y$ and $f \otimes A\colon X \times A \to_T Y \times A$ are defined to be $\mathbf{Set}$-arrows

$$A \otimes f\colon\ A \times X \xrightarrow{A \times f} A \times TY \xrightarrow{\mathrm{st}_{A,Y}} T(A \times Y)$$

$$f \otimes A\colon\ X \times A \xrightarrow{f \times A} TY \times A \xrightarrow{\mathrm{st}'_{Y,A}} T(Y \times A) \quad .$$

We write $f \otimes g$ for $(Y \otimes g) \circ_T (f \otimes Z)\colon X \times Z \to_T Y \times W$ if $(Y \otimes g) \circ_T (f \otimes Z) = (f \otimes W) \circ_T (X \otimes g)$ holds, for each pair of $\mathbf{Set}_T$-arrows $f\colon X \to_T Y$ and $g\colon Z \to_T W$. If $T$ is commutative, the premonoidal structure $\otimes$ gives monoidal products $(\otimes, 1)$ in this way, however this is not always the case in our setting.

In order to develop the mGoI framework and extend it to recursion, we require a monad $T$ on $\mathbf{Set}$ to make its Kleisli category $\mathbf{Set}_T$ satisfy some domain-theoretic properties.

**Requirement 2.2.1.** In our framework a monad $T$ on $\mathbf{Set}$ is required to satisfy the following conditions.

(a) The Kleisli category $\mathbf{Set}_T$ is $\mathbf{Cppo}$-enriched, i.e.

- every homset $\mathbf{Set}_T(X, Y)$ is an $\omega$-cppo with the least element $\perp$ and
- composition $\circ_T$ of $\mathbf{Set}_T$ are continuous.

(b) Composition $\circ_T$ 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 $f \circ_T \perp = \perp$ and $\perp \circ_T g^* = \perp$.

(c) The finite coproducts $(+, \emptyset)$ of $\mathbf{Set}_T$ is $\mathbf{Cppo}$-enriched, that is, cotupling $[-,-]_T$ of $\mathbf{Set}_T$ is continuous.

(d) 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 (a)–(c) above are in fact the sufficient conditions given in [18] to develop the mGoI framework, and we add the last condition (d) to accommodate recursion.

**Example 2.2.2.** Here are our leading examples of monads that satisfy Requirement 2.2.1. Note that all the monads given below are equipped with partiality, that is often obtained by adding $1 = \{*\}$. The partiality induces an $\omega$-cppo structure of each set $TX$ and hence gives a $\mathbf{Cppo}$-enrichment of the category $\mathbf{Set}_T$ in the pointwise manner.

- The *exception* monad $\mathcal{E}X = 1 + E + X$ for a set $E$.
- The *powerset* monad $\mathcal{P}X = \{U \subseteq X\}$.
- The *subdistribution* monad $\mathcal{D}X = \{d\colon X \to [0,1] \mid \sum_{x \in X} d(x) \leq 1\}$.
- 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 $TX = \mu Z.\,(1 + O \times Z + Z^I + X)$ for sets $I$ and $O$. By regarding sets $I$ and $O$ as $\omega$-cpo's with discrete orders, for any $\omega$-cppo $Z$, $F_X Z := 1 + O \times Z + Z^I + X$ becomes an $\omega$-cppo with the least element $* \in 1$. Hence $F_X$ is an endofunctor on $\mathbf{Cppo}$ and it has a final coalgebra; the I/O monad sends a set $X$ to the carrier of the final $F_X$-coalgebra in $\mathbf{Cppo}$.

It is shown in [18] that the conditions (a)–(c) in Requirement 2.2.1 induce a trace operator in the Kleisli category $\mathbf{Set}_T$.

**Lemma 2.2.3** ([18, Lemma 4.3])**.** If a monad $T$ on $\mathbf{Set}$ satisfies conditions (a)–(c) in Requirement 2.2.1, the Kleisli category $\mathbf{Set}_T$ has a trace operator $\mathrm{tr}^Z_{X,Y}\colon \mathbf{Set}_T(X + Z, Y + Z) \to \mathbf{Set}_T(X, Y)$ that is uniform in the following restricted form [14]: for any $\mathbf{Set}_T$-arrows $f\colon X + Z \to_T Y + Z$ and $g\colon X + W \to_T Y + W$ and $\mathbf{Set}$-arrow $h\colon Z \to W$, $(Y + h^*) \circ_T f = g \circ_T (X + h^*)$ implies $\mathrm{tr}^Z_{X,Y}(f) = \mathrm{tr}^W_{X,Y}(g)$. $\square$

### 2.2.2 Algebraic Operations on Monads

In [29] Plotkin and Power model algebraic signatures by families of arrows, called *algebraic operations*, using monads. We adopt an equivalent definition of algebraic operations given in [30].

**Definition 2.2.4** (algebraic operation on a monad [30])**.** Let $\mathbb{C}$ be a category with a cartesian closed structure $(\times, 1, \Rightarrow)$ and a monad $M$ on it, and $n$ be a natural number. A family $\{\alpha_{X,Y} \colon (X \Rightarrow MY)^n \to X \Rightarrow MY\}_{X \in \mathbb{C}^{\mathrm{op}}, Y \in \mathbb{C}_M}$ of $\mathbb{C}$-arrows is an *$n$-ary algebraic operation on $M$* if it is natural in $X \in \mathbb{C}^{\mathrm{op}}$ and $Y \in \mathbb{C}_M$, i.e. the following diagrams in $\mathbb{C}$ commute for any objects $X'$, $X$, $Y$ and $Y'$ in $\mathbb{C}$.

$$\begin{array}{ccc} (X' \Rightarrow X) \times (X \Rightarrow MY)^n & \xrightarrow{(X' \Rightarrow X) \times \alpha_{X,Y}} & (X' \Rightarrow X) \times (X \Rightarrow MY) \\ {\scriptstyle \langle \mathrm{comp} \circ ((X' \Rightarrow X) \times \pi_i) \rangle_{i=1}^n} \downarrow & & \downarrow {\scriptstyle \mathrm{comp}} \\ (X' \Rightarrow MY)^n & \xrightarrow[\alpha_{X',Y}]{} & X' \Rightarrow MY \end{array}$$

$$\begin{array}{ccc} (X \Rightarrow MY)^n \times (Y \Rightarrow MY') & \xrightarrow{\alpha_{X,Y} \times (Y \Rightarrow MY')} & (X \Rightarrow MY) \times (Y \Rightarrow MY') \\ {\scriptstyle \langle \mathrm{comp}_M \circ (\pi_i \times (Y \Rightarrow MY')) \rangle_{i=1}^n} \downarrow & & \downarrow {\scriptstyle \mathrm{comp}_M} \\ (X \Rightarrow MY')^n & \xrightarrow[\alpha_{X,Y'}]{} & X \Rightarrow MY' \end{array}$$

Here $(-)^n$ gives the $n$-fold product, $\pi_i$ is the $i$-th projection, $\langle - \rangle_{i=1}^n$ is the tupling, comp is composition of $\mathbb{C}$ and $\mathrm{comp}_M$ is that of $\mathbb{C}_M$.

For a monad $T$ on the particular category **Set**, an algebraic operation on $T$ can be given by a family of maps between homsets of **Set** (i.e. sets of **Set**-arrows).

**Example 2.2.5.** Here are algebraic operations on some of the monads listed in Example 2.2.2.

- For a set $E$ and each element $e \in E$, a 0-ary algebraic operation $\mathit{raise}_e$ on the exception monad $\mathcal{E}$ is equivalently given by the family $\{Y \to 1{+}E{+}Y\}_{Y \in \mathbf{Set}}$ of constant functions that always return $e$.

- A 2-ary algebraic operation $\oplus$ on the powerset monad $\mathcal{P}$ performs *non-deterministic choice.* For two functions $f, g \colon X \to \mathcal{P}Y$ it takes pointwise unions: $(f \oplus g)(x) = f(x) \cup g(x)$.

- 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 $f, g \colon X \to \mathcal{D}Y$ it superposes distributions in the pointwise manner: $(f \oplus_p g)(x)(y) = p \times f(x)(y) + (1 - p) \times g(x)(y)$.

- Let *Val* be a finite set and *Loc* be a set. For their elements $v \in \mathit{Val}$ and $\ell \in \mathit{Loc}$, a $|\mathit{Val}|$-ary algebraic operation $\mathit{lookup}_\ell$ and an 1-ary algebraic operation $\mathit{update}_{\ell,v}$ on the global state monad $\mathcal{S}X = (1 + X \times \mathit{Val}^{\mathit{Loc}})^{\mathit{Val}^{\mathit{Loc}}}$ perform *actions on global states.* For a family $\{f_v \colon X \to \mathcal{S}Y\}_{v \in \mathit{Val}}$ of functions, the former operation looks up global states $\sigma \in \mathit{Val}^{\mathit{Loc}}$; and for a function $f \colon X \to \mathcal{S}Y$, the latter operation updates global states:

$$\begin{aligned} \mathit{lookup}_\ell(\{f_v\}_{v \in \mathit{Val}})(x)(\sigma) &= f_{\sigma(\ell)}(x)(\sigma) \\ \mathit{update}_{\ell,v}(f)(x)(\sigma) &= f(x)(\sigma[\ell \mapsto v]) \end{aligned}$$

  where $\sigma[\ell \mapsto v](\ell')$ is equal to $v$ if $\ell' = \ell$ and to $\sigma(\ell')$ otherwise.

### 2.2.3 Algebraic Signatures Supported by Monads

Finally we characterize algebraic signatures that can be used to specify our target language, by looking at what monads can *support* the signatures.

**Definition 2.2.6.** We say a monad $T$ on **Set** *supports* an algebraic signature $\Sigma$ if, for each operation $\mathtt{op} \in \Sigma$, it has an $\mathrm{ar}(\mathtt{op})$-ary algebraic operation $op$ on $T$.

Our framework can translate terms in the language $\mathcal{L}_\Sigma$ if the algebraic signature $\Sigma$ is supported by a monad subject to Requirement 2.2.1.

All the algebraic signatures $\Sigma$ listed in Example 2.1.1 are indeed supported by monads $T$ listed in Example 2.2.2. The following table shows the correspondence between operations in $\Sigma$ and algebraic operations on $T$ listed in Example 2.2.5.

| algebraic signature $\Sigma$ | operation $\mathtt{op}$ | monad $T$ | algebraic operation $op$ |
|---|---|---|---|
| $\Sigma_{\mathrm{except}}$ | $\mathtt{raise}_e$ | $\mathcal{E}$ | $raise_e$ |
| $\Sigma_{\mathrm{nondet}}$ | $\mathtt{choose}$ | $\mathcal{P}$ | $\oplus$ |
| $\Sigma_{\mathrm{prob}}$ | $\mathtt{choose}_p$ | $\mathcal{D}$ | $\oplus_p$ |
| $\Sigma_{\mathrm{glstate}}$ | $\mathtt{lookup}_\ell$<br>$\mathtt{update}_{\ell,v}$ | $\mathcal{S}$ | $lookup_\ell$<br>$update_{\ell,v}$ |

Table 2.1: Examples of Algebraic Signatures Supported by Monads

# Chapter 3

# Component Calculus over Transducers

## 3.1 Categorical Geometry of Interaction

Before describing the component calculus, we briefly recall what we shall call *categorical GoI*, that is, the categorical formalization of GoI provided by Abramsky, Haghverdi and Scott in [1]. In the mGoI framework, the component calculus over transducers is developed with the intention of running machinery of categorical GoI with transducers.

What plays a crucial role in categorical GoI is a *GoI situation*, that is a particular traced symmetric monoidal category with an endofunctor accompanied by retractions.

**Definition 3.1.1** (traced symmetric monoidal category)**.** A category $\mathbb{C}$ is *traced symmetric monoidal* if it comes with:

- symmetric monoidal products $(\otimes, I)$, i.e. a bifunctor $\otimes\colon \mathbb{C}\times\mathbb{C}\to\mathbb{C}$ and an object $I\in\mathbb{C}$ together with the following four natural isomorphisms, and

$$\begin{aligned} a_{X,Y,Z}&\colon (X\otimes Y)\otimes Z \overset{\cong}{\Rightarrow} X\otimes(Y\otimes Z)\\ l_X&\colon I\otimes X \overset{\cong}{\Rightarrow} X\\ r_X&\colon X\otimes I \overset{\cong}{\Rightarrow} X\\ s_{X,Y}&\colon X\otimes Y \overset{\cong}{\Rightarrow} Y\otimes X \end{aligned}$$

- a trace operator $\mathrm{tr}^Z_{X,Y}\colon \mathbb{C}(X\otimes Z, Y\otimes Z)\to\mathbb{C}(X,Y)$

subject to certain coherence conditions (see e.g. [15, 19]).

**Definition 3.1.2** (traced symmetric monoidal functor)**.** Let $(\mathbb{C},\otimes,I,\mathrm{tr})$ be a traced symmetric monoidal category. A functor $F\colon\mathbb{C}\to\mathbb{C}$ is *traced symmetric monoidal* if it is:

- symmetric monoidal, i.e. equipped with an isomorphism $i\colon I\overset{\cong}{\Rightarrow} FI$ in $\mathbb{C}$ and a natural isomorphism

  $$m_{X,Y}\colon FX\otimes FY \overset{\cong}{\Rightarrow} F(X\otimes Y)$$

  subject to certain compatibility conditions (see e.g. [15, 19]) with four natural isomorphisms listed in Definition 3.1.1, and

- traced, i.e. subject to the equation

  $$\mathrm{tr}^{FZ}_{FX,FY}(m^{-1}_{Y,Z}\circ Ff\circ m_{X,Z}) = F(\mathrm{tr}^Z_{X,Y}(f))$$

  for any $\mathbb{C}$-arrow $f\colon X\otimes Z\to Y\otimes Z$.

**Definition 3.1.3** (retraction)**.** In a category $\mathbb{C}$, a *retraction* $f : X \lhd Y : g$ is a pair of $\mathbb{C}$-arrows $f\colon X \to Y$ and $g\colon Y \to X$ such that $g \circ f = \mathrm{id}_X$.

Recall that GoI gives token machine semantics of programs, as studied e.g. in [24]. A GoI situation is intuitively a category that can represent token machines, with various data carried by tokens, as well as constructions of token machines and axioms. It would be clear how a GoI situation represents all of them, in the following sections where we give constructions of transducers that form a concrete GoI situation.

We adapt the original definition of GoI situations given in [1] by slightly relaxing conditions of retractions in the same way as in [18, Remark 2.5]. Monoidal naturality of pairs $(d, d')$ and $(c, c')$ is crucial to prove Theorem 3.3.8.

**Definition 3.1.4** (GoI situation)**.** A *GoI situation* is a list $((\mathbb{C}, \otimes, I, \mathrm{tr}), F, U, \phi, \psi, u, v)$ that consists of a traced symmetric monoidal category $(\mathbb{C}, \otimes, I, \mathrm{tr})$, a traced symmetric monoidal functor $F\colon \mathbb{C} \to \mathbb{C}$, an object $U \in \mathbb{C}$ and the following retractions in $\mathbb{C}$:

$$\begin{aligned}
\phi &: U \otimes U \lhd U : \psi \\
u &: FU \lhd U : v \\
n &: I \lhd U : n' \\
e_X &: X \lhd FX : e'_X && \text{(dereliction)} \\
d_X &: FFX \lhd FX : d'_X && \text{(digging)} \\
c_X &: FX \otimes FX \lhd FX : c'_X && \text{(contraction)} \\
w_X &: I \lhd FX : w'_X && \text{(weakening)}
\end{aligned}$$

such that $e_X, w_X$ are natural in $X \in \mathbb{C}$ and $d_X, d'_X, c_X, c'_X$ are monoidal natural in $X \in \mathbb{C}$.

Categorical GoI captures how GoI gives an interpretation of programs, by providing a way to obtain a *linear combinatory algebra* from a GoI situation. Further combined with the Girard translation of linear logic to intuitionistic logic, categorical GoI can yield an *SK-algebra* that is a model of untyped $\lambda$-calculus.

**Proposition 3.1.5.** Let $((\mathbb{C}, \otimes, I, \mathrm{tr}), F, U, \phi, \psi, u, v)$ be a GoI situation. The homset $\mathbb{C}(U, U)$ is an SK-algebra, with a binary operation $\cdot\colon \mathbb{C}(U, U)^2 \to \mathbb{C}(U, U)$ defined by

$$f \cdot g := \mathrm{tr}^U_{U,U}(\psi \circ f \circ \phi \circ (U \otimes (u \circ Fg \circ v)))$$

and two elements $\mathsf{S}, \mathsf{K} \in \mathbb{C}(U, U)$ that satisfy

$$\mathsf{S} \cdot f \cdot g \cdot h = f \cdot h \cdot (g \cdot h), \qquad \mathsf{K} \cdot f \cdot g = f$$

for any $f, g, h \in \mathbb{C}(U, U)$, where $\cdot$ is assumed to be left associative. □

### 3.2 Component Calculus in the mGoI Framework

This section recalls the component calculus over transducers developed in the mGoI framework [18, Section 4.2]. The component calculus intends to provide a GoI situation that yields an SK-algebra with extra operations on it via categorical GoI, so that the extra operations can be exploited to interpret algebraic effects.

Transducers are precisely $T$-transducers, with a monad $T$ on **Set** that models computational effects. They can be thought of as "effectful" Mealy machines and, in terms of GoI, as "effectful and memoryful" token machines.

**Definition 3.2.1** ($T$-transducer [18, Definition 4.1])**.** For sets $A$ and $B$, a $T$-*transducer* $(X, c, x)$ *from* $A$ *to* $B$ consists of a set $X$, a $\mathbf{Set}_T$-arrow $c\colon X \times A \to_T X \times B$ and a $\mathbf{Set}$-arrow $x\colon 1 \to X$.

We write $(X, c, x)\colon A \rightharpoondown B$ if a $T$-transducer $(X, c, x)$ is from $A$ to $B$. It is intuitively an "effectful" transition function $c\colon X \times A \to T(X \times B)$ with a set of input $A$, a set of output $B$, a state space $X$ and an initial state $x \in X$. Given input in $A$, the transition function $c$ internally updates its internal state (or "memory") and generates output in $B$. On this intuition we depict a $T$-transducer $(X, c, x)\colon A \rightharpoondown B$ as in Figure 3.1.

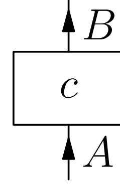



Figure 3.1: a $T$-transducer $(X, c, x)\colon A \rightharpoondown B$

In a transition function $c\colon X \times A \to T(X \times B)$, a monad $T$ is used to model computational effects. For example, if $T$ is the powerset monad $\mathcal{P}$ the transition function $c$ is nondeterministic and returns a set of possible output, and if $T$ is a global state monad $\mathcal{S}$ the transition function generates output according to the stored values of global states (see Example 2.2.2 for the monads $\mathcal{P}$ and $\mathcal{S}$).

A $T$-transducer can be "memoryless." A $\mathbf{Set}_T$-arrow $f\colon A \to_T B$ is lifted to a $T$-transducer

$$J(f) := (1,\ 1 \times A \overset{\cong}{\Rightarrow} A \overset{f}{\to} TB \overset{\cong}{\Rightarrow} T(1 \times B),\ \mathrm{id}_1) : A \rightharpoondown B$$

that performs the same computation as $f$ without internal states. This construction $J$ of $T$-transducers from $\mathbf{Set}_T$-arrows can be combined with the lifting $(-)^*$ from $\mathbf{Set}$-arrows to $\mathbf{Set}_T$-arrows, that yields "memoryless and pure" $T$-transducers. A $\mathbf{Set}$-arrow $g\colon A \to B$ performs pure (i.e. non-effectful) computation and is lifted to a $T$-transducer $J(g^*)\colon A \rightharpoondown B$.

### 3.2.1 Operators on Transducers

The component calculus over $T$-transducers, developed in the mGoI framework, is comprised of primitive "memoryless" $T$-transducers and the following operators on $T$-transducers: (a) sequential composition $\circ$, (b) binary parallel composition $\boxplus$, (c) the trace operator Tr, (d) the countable copy operator $F$, (e) binary application $\bullet$, and (f) lifted algebraic operations $\overline{\alpha}$. Figure 3.2 depicts how these operators act on $T$-transducers. We describe the operators first and give primitives later.

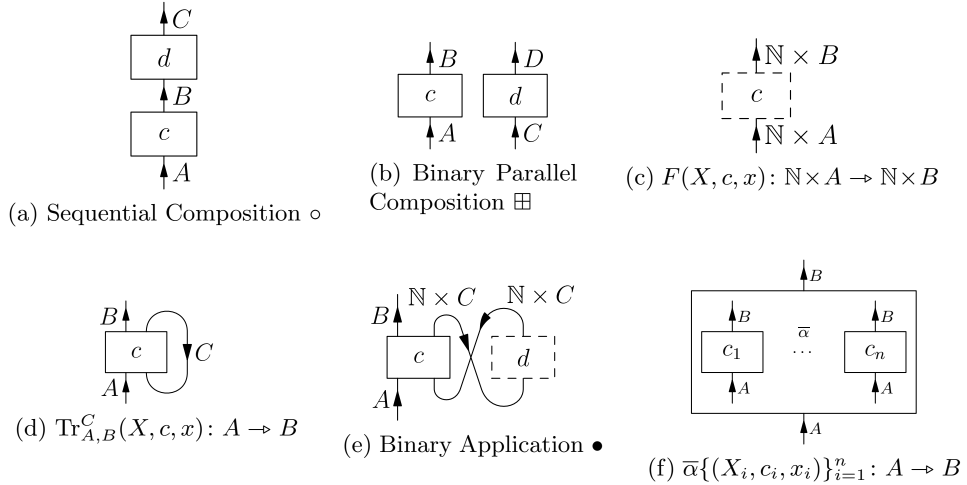



Figure 3.2: Operators on $T$-transducers

**Sequential and Binary Parallel Composition** $\circ, \boxplus$ Two $T$-transducers can be composed in two different ways. One is *sequential*: two $T$-transducers are executed one by one, in which output of one $T$-transducer is passed to input of the other $T$-transducer. The other composition is *parallel*: one of two $T$-transducers is chosen and executed according to input. This would be more clear in the depictions (a) and (b) in Figure 3.2.

**Definition 3.2.2** (sequential composition $\circ$)**.** *Sequential composition* $(Y, d, y) \circ (X, c, x)\colon A \to C$ of two $T$-transducers $(X, c, x)\colon A \to B$ and $(Y, d, y)\colon B \to C$ is defined to be

$$\Big( X \times Y \;,\quad X \times Y \times A \xrightarrow{X \otimes \sigma^*_{Y,A}}_T X \times A \times Y \xrightarrow{c \otimes Y}_T X \times B \times Y \\ \xrightarrow{X \otimes \sigma^*_{B,Y}}_T X \times Y \times B \xrightarrow{X \otimes d}_T X \times Y \times C \quad , \quad \langle x, y \rangle \Big) \; .$$

**Definition 3.2.3** (binary parallel composition $\boxplus$)**.** *Binary parallel composition* $(X, c, x) \boxplus (Y, d, y)\colon A + C \to B + D$ of two $T$-transducers $(X, c, x)\colon A \to B$ and $(Y, d, y)\colon B \to C$ is defined to be $(X \times Y, e, \langle x, y \rangle)$ where $e$ is the unique $\mathbf{Set}_T$-arrow that makes the following two diagrams in $\mathbf{Set}_T$ commute.

$$\begin{array}{ccccc}
X \times Y \times A & \xrightarrow{X \otimes (Y \otimes \mathrm{inl}^*)} & X \times Y \times (A + C) & \xrightarrow{e} & X \times Y \times (B + D) \\
\downarrow{\scriptstyle \sigma^*_{X,Y} \otimes A} & & & & \uparrow{\scriptstyle \sigma^*_{Y,X} \otimes \mathrm{inl}^*} \\
Y \times X \times A & & \xrightarrow[Y \otimes c]{} & & Y \times X \times B
\end{array}$$

$$\begin{array}{ccccc}
X \times Y \times C & \xrightarrow{X \otimes (Y \otimes \mathrm{inr}^*)} & X \times Y \times (A + C) & \xrightarrow{e} & X \times Y \times (B + D) \\
& \searrow{\scriptstyle X \otimes d} & & \nearrow{\scriptstyle X \otimes (Y \otimes \mathrm{inr}^*)} & \\
& & X \times Y \times D & &
\end{array}$$

In the above definitions we utilize the bicartesian structure of **Set**, namely tupling $\langle -, - \rangle$, swapping $\sigma_{X,Y}\colon X \times Y \xrightarrow{\cong} Y \times X$ and injections $Z \xrightarrow{\mathrm{inl}} Z + W \xleftarrow{\mathrm{inr}} W$.

**Countable Copy Operator** $F$ We can make a bunch of countably many copies of a $T$-transducer, that is simply depicted by a dashed box in Figure 3.2 (c). Input to a bunch of copies includes an index number that determines which copy to be executed, and output from a bunch of copies also includes an index number that indicates which copy has been executed. An index number is given by a natural number; recall that $\mathbb{N}$ is the set of natural numbers.

**Definition 3.2.4** (countable copy operator $F$)**.** For a $T$-transducer $(X, c, x)\colon A \to B$, the $T$-transducer $F(X, c, x)\colon \mathbb{N} \times A \to \mathbb{N} \times B$ is defined to be $(X^{\mathbb{N}}, e, \langle x \rangle_{i \in \mathbb{N}})$ where $e$ is the unique $\mathbf{Set}_T$-arrow that makes the following diagram in $\mathbf{Set}_T$ commutes for each $i \in \mathbb{N}$.

$$\begin{array}{ccccc}
X^{\mathbb{N}} \times A & \xrightarrow{X^{\mathbb{N}} \otimes \mathrm{inj}_i^*} & X^{\mathbb{N}} \times (\mathbb{N} \times A) & \xrightarrow{e} & X^{\mathbb{N}} \times (\mathbb{N} \times B) \\
\downarrow{\scriptstyle \sigma_i^* \otimes A} & & & & \uparrow{\scriptstyle (\sigma_i^{-1})^* \otimes \mathrm{inj}_i^*} \\
X^{\mathbb{N}} \times X \times A & & \xrightarrow[X^{\mathbb{N}} \otimes c]{} & & X^{\mathbb{N}} \times X \times B
\end{array}$$

In this definition we utilize the *infinite* bicartesian structure of **Set**, especially $\mathbb{N}$-fold coproduct $\mathbb{N} \times (-)$ with the $i$-th injection $Z \xrightarrow{\mathrm{inj}_i} \mathbb{N} \times Z$, and $\mathbb{N}$-fold

product $(-)^{\mathbb{N}}$ with tupling $\langle - \rangle_{i\in\mathbb{N}}$ and swapping $\sigma_i \colon X^{\mathbb{N}} \overset{\cong}{\Rightarrow} X^{\mathbb{N}} \times X$. Precisely the swapping $\sigma_i$ picks the $i$-th element of a given infinite list as below.

$$\sigma_i(x_0, \dots, x_{i-1}, x_i, x_{i+1}, \dots) = ((x_0, \dots, x_{i-1}, x_{i+1}, \dots), x_i)$$

**Trace Operator** Tr Output of some $T$-transducer can be fed back to input of the $T$-transducer itself. The trace operator Tr performs this construction, namely making a "self-feedback loop" as in Figure 3.2 (d).

**Definition 3.2.5** (trace operator Tr)**.** *Trace* $\mathrm{Tr}^{C}_{A,B}(X, c, x) \colon A \multimap B$ of a $T$-transducer $(X, c, x) \colon A + C \multimap B + C$ is defined to be $(X, \mathrm{tr}^{X\times C}_{X\times A, X\times B}(c'), x)$ where $c'$ is the following $\mathbf{Set}_T$-arrow:

$$\begin{aligned} X \times A + X \times C &\xrightarrow{(\delta^{-1}_{X,A,C})^*}_T X \times (A + C) \xrightarrow{c}_T X \times (B + C) \\ &\xrightarrow{\delta^*_{X,B,C}}_T X \times A + X \times C \enspace . \end{aligned}$$

As stated in Lemma 2.2.3, a monad $T$ subject to Requirement 2.2.1 induces a trace operator tr of $\mathbf{Set}_T$. We apply it to the composition of $c$ and distribution $\delta_{X,Y,Z} \colon X \times (Y + Z) \overset{\cong}{\Rightarrow} X \times Y + X \times Z$ in $\mathbf{Set}$.

**Binary Application** $\bullet$ Combining all the operators described above, two $T$-transducers can be "composed" in another way that corresponds to the game-theoretic *parallel composition and hiding* construction and is used to translate function applications.

**Definition 3.2.6** (binary application $\bullet$)**.** *Binary application* $(X, c, x) \bullet (Y, d, y) \colon A \multimap B$ of two $T$-transducers $(X, c, x) \colon A + \mathbb{N} \times C \multimap B + \mathbb{N} \times C$ and $(Y, d, y) \colon C \multimap C$ is defined to be

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

The depiction (e) in Figure 3.2 can be obtained by chasing the depiction of the definition above as a graph. This operator $\bullet$ is an adaptation of the operator $\bullet$ defined in [18, Section 4.2.4].

**Lifted Algebraic Operations** $\overline{\alpha}$ Each algebraic operation on $T$ can be lifted to an operator on $T$-transducers.

**Definition 3.2.7** (lifted algebraic operation $\overline{\alpha}$)**.** Let $\alpha$ be an $n$-ary algebraic operation on $T$. For a family $\{(X_i, c_i, x_i) \colon A \multimap B\}^n_{i=1}$ of $T$-transducers, the $T$-transducer $\overline{\alpha}\{(X_i, c_i, x_i)\}^n_{i=1} \colon A \multimap B$ is defined to be $(1 + \coprod^n_{i=1} X_i, e, \mathrm{inl}^*)$ where $e$ is the unique $\mathbf{Set}_T$-arrow that makes the following diagrams in $\mathbf{Set}_T$ commute

for each $j = 1, \ldots, n$.

$$\begin{array}{ccccc}
1 \times A & \xrightarrow{\mathrm{inl}^* \otimes A} & (1 + \coprod_{i=1}^{n} X_i) \times A & \xrightarrow{e} & (1 + \coprod_{i=1}^{n} X_i) \times B \\
\downarrow{\scriptstyle x_j^* \otimes A} & & & & \uparrow{\scriptstyle (\mathrm{inr} \circ \mathrm{inj}_j)^* \otimes B} \\
X_j \times A & & \xrightarrow[c_j]{} & & X_j \times B
\end{array}$$

$$\begin{array}{ccccc}
X_j \times A & \xrightarrow{(\mathrm{inr} \circ \mathrm{inj}_j)^* \otimes A} & (1 + \coprod_{i=1}^{n} X_i) \times A & \xrightarrow{e} & (1 + \coprod_{i=1}^{n} X_i) \times B \\
& \searrow^{c_j} & & \nearrow_{(\mathrm{inr} \circ \mathrm{inj}_j)^* \otimes B} & \\
& & X_j \times B & &
\end{array}$$

Given a family $\{(X_i, c_i, x_i)\}_{i=1}^{n}$ of $T$-transducers, the operator $\overline{\alpha}$ just introduces a fresh initial state $* \in 1$ and "effectful" transitions from the state $*$ to each initial state $x_i$ of the given $T$-transducer $(X_i, c_i, x_i)$. After making the "effectful" first transition from the initial state $*$ to a state $x_i$, the resulting $T$-transducer $\overline{\alpha}\{(X_i, c_i, x_i)\}_{i=1}^{n}$ memorizes the result of the transition by its internal states (or "memories") and behaves exactly as the $T$-transducer $(X_i, c_i, x_i)$.

### 3.2.2 Primitive "Memoryless" Transducers

Primitive $T$-transducers of mGoI's component calculus are all "memoryless," i.e. lifted from $\mathbf{Set}_T$-arrows by the construction $J$. Below we give their underlying $\mathbf{Set}_T$-arrows, or sometimes $\mathbf{Set}$-arrows that can be made into $\mathbf{Set}_T$-arrows via the lifting $(-)^*$.

One class of the underlying arrows are in the form of retractions. We use the two chosen bijections

$$\phi : \mathbb{N} + \mathbb{N} \cong \mathbb{N} : \psi \qquad\qquad \mathrm{u} : F\mathbb{N} \cong \mathbb{N} : \mathrm{v} \tag{3.1}$$

in $\mathbf{Set}$, and four retractions

$$\begin{array}{lr}
\tilde{\mathrm{e}}_A : A \lhd FA : \tilde{\mathrm{e}'}_A & \text{(dereliction)} \\
\tilde{\mathrm{d}}_A : FFA \cong FA : \tilde{\mathrm{d}'}_A & \text{(digging)} \\
\tilde{\mathrm{c}}_A : FA + FA \cong FA : \tilde{\mathrm{c}'}_A & \text{(contraction)} \\
\tilde{\mathrm{w}}_A : \emptyset \lhd FA : \tilde{\mathrm{w}'}_A & \text{(weakening)}
\end{array} \;, \tag{3.2}$$

the first three of which are in $\mathbf{Set}$ and the last one of which is in $\mathbf{Set}_T$. The four retractions are defined by

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

For each set $X$, a set $FX$ is defined to be $\mathbb{N} \times X$ and a $\mathbf{Set}_T$-arrow $!_X : \emptyset \to_T X$ is the unique arrow from the initial object $\emptyset$.

As suggested by the abuse of notation, these retractions are related to the countable copy operator $F$. We can use the four retractions in (3.2) to manipulate a bunch of copies of a $T$-transducer that is generated by the operator $F$, namely by pre-composing the injection side and post-composing the surjection side to a $T$-transducer in the form of $F(X, c, x)$. Given a bunch of copies of a $T$-transducer, dereliction picks one copy, digging sorts it to "a bunch of bunches of copies," contraction splits it to two bunches of copies and weakening totally discards it, as illustrated in [18, Section 4.2.4]. Technically this is due to naturality of $\tilde{\mathrm{e}}_A, \tilde{\mathrm{d}}_A, \tilde{\mathrm{c}}_A$ in $A \in \mathbf{Set}$ and that of $\tilde{\mathrm{w}}_A$ in $A \in \mathbf{Set}_T$.

In the concrete translation of terms to transducers given in the next chapter, we will use four "suppressed" retractions

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

that can be obtained by composing bijections in (3.1) and retractions in (3.2). They are precisely defined by

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

We use the *dynamic algebra* notation by writing $\mathtt{g}$ for $\phi \circ \mathrm{inl}$, $\mathtt{d}$ for $\phi \circ \mathrm{inr}$ and $\langle -, - \rangle$ for $\mathrm{u}(-, -)$.[1]

The other class of underlying arrows of primitive $T$-transducers includes the following **Set**-arrows

$$\begin{aligned}
\mathrm{k}_n \colon \mathbb{N} \to \mathbb{N} & \qquad\qquad \mathrm{h} \colon \mathbb{N} + \mathbb{N} \to \mathbb{N} + \mathbb{N} \\
\mathrm{sum} \colon \mathbb{N} + \mathbb{N} + \mathbb{N} \to \mathbb{N} + \mathbb{N} + \mathbb{N} &
\end{aligned}$$

defined concretely by

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

In the translation of terms to transducers, $\mathrm{k}_n$ is used to translate the constant term $\underline{n}$, sum is used to translate the arithmetic primitive $+$, and h is used to "force" the call-by-value evaluation strategy by serving as a CPS-like construct.

[1] $\mathtt{g}$ is for *left* and $\mathtt{d}$ is for *right*, in French. See e.g. [27, 21].

### 3.2.3 "Category" of Transducers

It would be natural to ask what axioms the operators on $T$-transducers satisfy. An answer to the question given in [18] is a categorical one, following observations in [17], in which reasoning on $T$-transducers is not equational but up to *behavioral equivalence.*

**Definition 3.2.8** (homomorphism between $T$-transducers [18, Definition 5.1])**.** Let $(X, c, x), (Y, d, y) \colon A \twoheadrightarrow B$ be $T$-transducers. A *homomorphism from* $(X, c, x)$ *to* $(Y, d, y)$ is a **Set**-arrow $f \colon X \to Y$ that makes both the left diagram below in $\mathbf{Set}_T$ and the right diagram below in **Set** commute.

$$\begin{array}{ccc} X \times B & \xrightarrow{h^* \otimes B} & Y \times B \\ {\scriptstyle c}\uparrow & & \uparrow{\scriptstyle d} \\ X \times A & \xrightarrow[h^* \otimes A]{} & Y \times A \end{array} \qquad \begin{array}{ccc} X & \xrightarrow{h} & Y \\ {\scriptstyle x}\uparrow & \nearrow{\scriptstyle y} & \\ 1 & & \end{array}$$

**Definition 3.2.9** (behavioral equivalence [18, Definition 5.2])**.** Two $T$-transducers $(X, c, x), (Y, d, y) \colon A \twoheadrightarrow B$ are *behavioral equivalent* if there exists a $T$-transducer $(Z, e, z) \colon A \twoheadrightarrow B$ and homomorphisms from $(X, c, x)$ to $(Z, e, z)$ and from $(Y, d, y)$ to $(Z, e, z)$.

We write $(X, c, x) \simeq (Y, d, y)$ if $(X, c, x)$ and $(Y, d, y)$ are behavioral equivalent.

Behavioral equivalence enables us to abstract away from state spaces of $T$-transducers. For example, a $T$-transducer $(X, c, x) \colon A \twoheadrightarrow B$ is not equal to a composed $T$-transducer $J(\mathrm{id}^*_B) \circ (X, c, x) \colon A \twoheadrightarrow B$ because their state spaces are not equal but isomorphic, namely $X$ and $X \times 1$ respectively. By choosing an isomorphism $X \overset{\cong}{\Rrightarrow} X \times 1$ as a homomorphism, the two $T$-transducers can be identified via behavioral equivalence.

All the operators on $T$-transducers introduced so far are compatible with behavioral equivalence, and axioms they satisfy can be described in the categorical term. We list the facts about the component calculus that are investigated in [18].

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

  is indeed a category and it has a traced symmetric monoidal structure $(\boxplus, \emptyset, \mathrm{Tr})$.

- The countable copy operator $F$, abused for sets as $FX := \mathbb{N} \times X$, is a traced symmetric monoidal functor on $(\mathbf{Res}(T), \boxplus, \emptyset, \mathrm{Tr})$. This fact enables us to identify a $T$-transducer $F(X, c, x) \colon F(A + C) \twoheadrightarrow F(B + D)$ with $F(X, c, x) \colon FA + FC \twoheadrightarrow FB + FD$ via behavioral equivalence, as done e.g. in Figure 3.3.

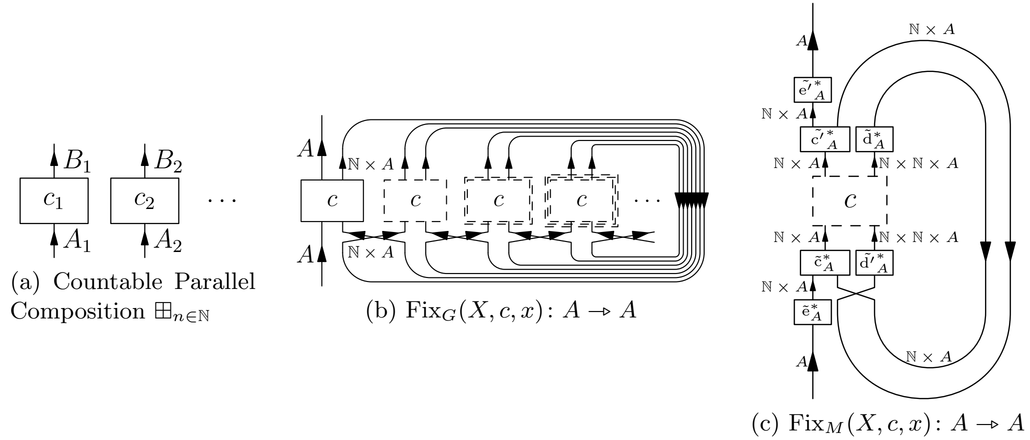
- The list $((\mathbf{Res}(T), \boxplus, \emptyset, \mathrm{Tr}), F, \mathbb{N}, J(\phi^*), J(\psi^*), J(\mathrm{u}^*), J(\mathrm{v}^*))$, together with primitive $T$-transducers lifted from the retractions in (3.2), forms a GoI situation (see Definition 3.1.4). This means naturality of $\tilde{\mathrm{e}}_A, \tilde{\mathrm{w}}_A$ lifts to that of $J(\tilde{\mathrm{e}}^*_A), J(\tilde{\mathrm{w}}_A)$, and additionally monoidal naturality of $\tilde{\mathrm{d}}_A, \tilde{\mathrm{d}'}_A, \tilde{\mathrm{c}}_A, \tilde{\mathrm{c}'}_A$ lifts to that of $J(\tilde{\mathrm{d}}^*_A), J(\tilde{\mathrm{d}'}^*_A), J(\tilde{\mathrm{c}}^*_A), J(\tilde{\mathrm{c}'}^*_A)$.

- [18, Theorem 5.3] Operators $\overline{\alpha}$ are natural and Tr distributes over them up to the behavioral equivalence.

Figure 3.1 & 3.2 can be therefore seen as string diagrams in the traced symmetric monoidal category $(\mathbf{Res}(T), \boxplus, \emptyset, \mathrm{Tr})$.

## 3.3 Extension of Component Calculus

In the last sections we explained the component calculus over $T$-transducers developed in the mGoI framework. We extend it by introducing new operators, namely two styles of *fixed point operators*, and an auxiliary operator, that is *countable* parallel composition, so that our framework can accommodate recursion. Figure 3.3 depicts how these operators act on $T$-transducers.




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

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

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

Figure 3.3: New Operators on $T$-transducers

Denotationally recursion is often interpreted as infinitely many self-applications of some function, and operationally it can be understood as infinitely many repeats of some procedure. Our new two fixed point operators, used in the concrete translation of recursion to transducers, are based on this intuition. One is *Girard style*, in which a given $T$-transducer is "applied" to itself for infinitely many times via binary application $\bullet$, and the other one is *Mackie style*, in which "self-feedback loops" are made for a given $T$-transducer via the trace operator Tr.

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

As an auxiliary operator on $T$-transducers, we define countable parallel composition $\boxplus_{n\in\mathbb{N}}$ by extending binary parallel composition $\boxplus$. This extension is possible because **Set** have both infinite products $\prod_{n\in\mathbb{N}}$ and coproducts $\coprod_{n\in\mathbb{N}}$ that are inherited by $\mathbf{Set}_T$.

**Definition 3.3.1** (countable parallel composition $\boxplus_{n\in\mathbb{N}}$). *Countable parallel composition* $\boxplus_{n\in\mathbb{N}}\{(X_n, c_n, x_n)\} \colon \coprod_{n\in\mathbb{N}} A_n \multimap \coprod_{n\in\mathbb{N}} B_n$ of a family $\{(X_n, c_n, x_n) \colon A_n \multimap B_n\}$ of $T$-transducers is defined to be $(\prod_{n\in\mathbb{N}} X_n, e, \langle x_n \rangle_{n\in\mathbb{N}})$ where $e$ is the unique $\mathbf{Set}_T$-arrow that makes the following diagram in $\mathbf{Set}_T$ commutes for each

$i \in \mathbb{N}$.

$$\begin{array}{ccccc}
(\prod_{n\in\mathbb{N}} X_n) \times A_i & \xrightarrow{(\prod_{n\in\mathbb{N}} X_n)\otimes \mathrm{inj}_i^*} & (\prod_{n\in\mathbb{N}} X_n) \times (\coprod_{n\in\mathbb{N}} A_n) & \xrightarrow{e} & (\prod_{n\in\mathbb{N}} X_n) \times (\coprod_{n\in\mathbb{N}} B_n) \\
\downarrow{\scriptstyle \sigma_i^* \otimes A_i} & & & & \uparrow{\scriptstyle (\sigma_i^{-1})^* \otimes \mathrm{inj}_i^*} \\
(\prod_{n\in\mathbb{N}} X_n) \times X_i \times A_i & & \xrightarrow[(\prod_{n\in\mathbb{N}} X_n)\otimes c_i]{} & & (\prod_{n\in\mathbb{N}} X_n) \times X_i \times B_i
\end{array}$$

It is easy to confirm that this operator is compatible with behavioral equivalence.

Countable parallel composition $\boxplus_{n\in\mathbb{N}}$ can be seen as a generalization of the countable copy operator $F$ as well. For any $T$-transducer $(X, c, x)$, we have the behavioral equivalence $\boxplus_{n\in\mathbb{N}}\{(X, c, x)\} \simeq F(X, c, x)$ that directly follows from Definition 3.2.4 and Definition 3.3.1.

### 3.3.2 Girard Style Fixed Point Operator $\mathrm{Fix}_G$

The Girard style fixed point operator $\mathrm{Fix}_G$ intends to produce infinitely many "self-binary-applications" of a given $T$-transducer.

**Definition 3.3.2** (Girard style fixed point operator $\mathrm{Fix}_G$)**.** For a $T$-transducer $(X, c, x)\colon A + \mathbb{N} \times A \mathrel{\ooalign{$\rightharpoonup$}} A + \mathbb{N} \times A$, the $T$-transducer $\mathrm{Fix}_G(X, c, x)\colon A \rightharpoonup A$ is defined to be

$$\mathrm{Tr}^N_{A,A}(\, \boxplus_{n\in\mathbb{N}}\{F^n(X, c, x)\} \circ J((\mathrm{id}_A + \coprod_{i\in\mathbb{N}} \sigma'_{N_i,N_i})^*)\,)$$

where $N$ is defined by

$$N := \coprod_{i\in\mathbb{N}} (N_i + N_i) \ , \qquad \begin{aligned} N_0 &:= \mathbb{N} \times A \\ N_1 &:= \mathbb{N} \times \mathbb{N} \times A \\ N_2 &:= \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times A \\ &\vdots \end{aligned}$$

and $\sigma'_{R,S}$ is swapping $R + S \overset{\cong}{\Rightarrow} S + R$ with respect to coproducts of **Set**.

Figure 3.3 (b) precisely depicts the above definition. By chasing the depiction as a graph, we can obtain another depiction of a $T$-transducer $\mathrm{Fix}_G(X, c, x)\colon A \rightharpoonup A$ shown in Figure 3.4, that can be seen as an infinitely long chain of binary applications (see also Figure 3.2 (e)).

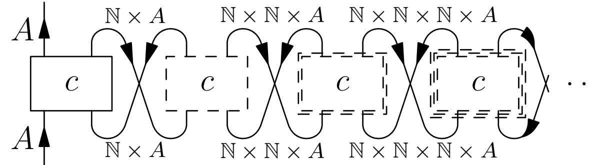


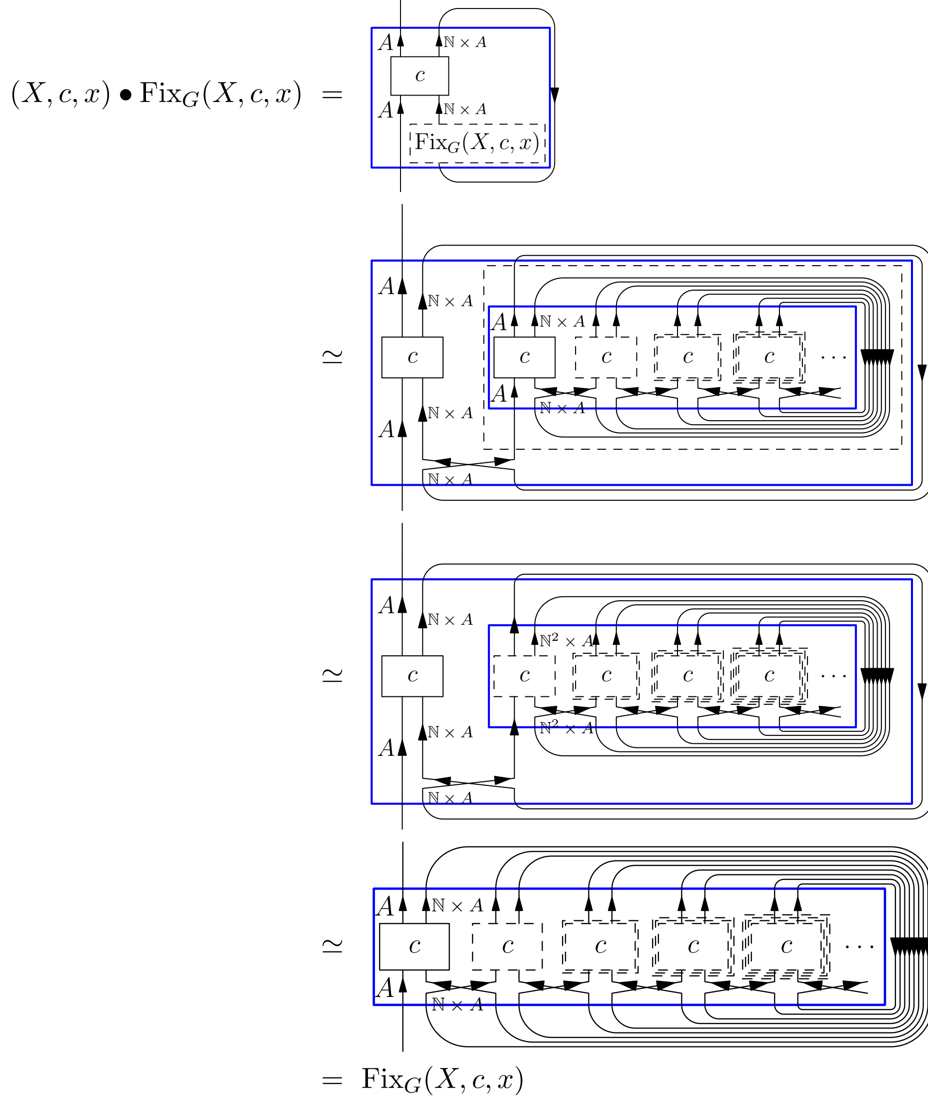

Figure 3.4: Another Depiction of $\mathrm{Fix}_G(X, c, x)\colon A \rightharpoonup A$

This infinitely long chain of binary applications in fact yields a fixed point with respect to binary application $\bullet$, that is why we call $\mathrm{Fix}_G$ "fixed point" operator.

**Proposition 3.3.3.** Let $(X, c, x)\colon A + \mathbb{N} \times A \rightharpoonup A + \mathbb{N} \times A$ be a $T$-transducer. The $T$-transducer $\mathrm{Fix}_G(X, c, x)\colon A \rightharpoonup A$ satisfies the following behavioral equivalence

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

*Proof.* We illustrate the proof using diagrammatic depictions below, where blue boxes indicate to which $T$-transducer the trace operator Tr is applied.




The first behavioral equivalence follows from trace axioms [15, 19] satisfied by the trace operator Tr and from naturality of swapping $\sigma'$. The second one exploits the fact, noted in Section 3.2.3, that the countable copy operator $F$ gives a traced symmetric monoidal functor on the category $\mathbf{Res}(T)$. The last one follows from trace axioms again, and additionally from the following behavioral equivalence

$$(Y_0, d_0, y_0) \boxplus (\boxplus_{n\in\mathbb{N}}\{(Y_{n+1}, d_{n+1}, y_{n+1})\}) \simeq \boxplus_{n\in\mathbb{N}}\{(Y_n, d_n, y_n)\}$$

that is easy to be confirmed for any family $\{(Y_n, d_n, y_n)\}_{n\in\mathbb{N}}$ of $T$-transducers. $\square$

### 3.3.3 Induced $\omega$-cpo Structure on Transducers

The Girard style fixed point operator $\mathrm{Fix}_G$ can be characterized not only by a fixed point but also by a limit of finite approximations. This means that the

operator $\mathrm{Fix}_G$ enables us to translate recursion as a limit of finite approximations, as much like Girard's approach [12] to interpret recursion in GoI. Therefore we call $\mathrm{Fix}_G$ "Girard style" fixed point operator.

Precisely the operator $\mathrm{Fix}_G$ yields a supremum with respect to an $\omega$-cpo structure on $T$-transducers. Since a monad $T$ is subject to Requirement 2.2.1, each homset of the category $\mathbf{Set}_T$ is an $\omega$-cppo $(\mathbf{Set}_T(Z,W),\sqsubseteq,\bot)$. This $\omega$-cppo enrichment of $\mathbf{Set}_T$ induces an $\omega$-cpo structure on $T$-transducers, namely an $\omega$-cpo $(\mathbf{Trans}(T)(A,B),\trianglelefteq)$ for each set $\mathbf{Trans}(T)(A,B)$ of $T$-transducers from $A$ to $B$, in the following way.

- A binary 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 $(X,c,x),(Y,d,y)\colon A \multimap B$, is a partial order.

- Let $(X,c_1,x) \trianglelefteq (X,c_2,x) \trianglelefteq \cdots$ be an $\omega$-chain of $T$-transducers from $A$ to $B$. Its supremum $\sup_{i\in\omega}(X,c_i,x)\colon A \multimap B$ is given by $(X,\sup_{i\in\omega} c_i,x)$.

- Additionally, for any set $X$ and $\mathbf{Set}$-arrow $x\colon 1 \to X$, a $T$-transducer $(X,\bot,x)\colon A \multimap B$ gives a minimal element.

This partial order $\trianglelefteq$ is quite "raw," in the sense that it is only defined when two $T$-transducers have the same state spaces and the same initial states. The order $\trianglelefteq$ forces us to stay aware of state spaces of $T$-transducers in domain-theoretic reasoning, while behavioral equivalence $\simeq$ enables us to abstract away from state spaces of $T$-transducers in equational reasoning. Currently we have no way to relate the order $\trianglelefteq$ to behavioral equivalence $\simeq$, however, in spite of this inconvenience, the order $\trianglelefteq$ has enough power to help us prove the adequacy result.

As the order $\trianglelefteq$ on $T$-transducers is "lifted" from the order $\sqsubseteq$ on $\mathbf{Set}_T$-arrows, our component calculus over $T$-transducers "inherits" domain-theoretic properties from the category $\mathbf{Set}_T$.

**Lemma 3.3.4.** 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 $\mathrm{Tr}$ | ✓ | ✓ |
| binary application $\bullet$ | ✓ | ✓$_{\star_3}$ |
| lifted algebraic operation $\overline{\alpha}$ | ✓ | × |
| countable parallel composition $\boxplus_{n\in\mathbb{N}}$ | ✓ | ✓$_{\star_2}$ |
| Girard style fixed point operator $\mathrm{Fix}_G$ | ✓ | ✓ |

$\star_1$ 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 any $T$-transducer $(Z,e^*,z)$ with its transition function 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 $(X, \bot, x)$ with its transition function $\bot$, 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 2.2.1. These conditions imply the facts about the category $\mathbf{Set}_T$ listed below, and once we observe them the properties of the component calculus can be easily confirmed by definitions of operators.

- Countable cotupling $[\{-\}_{n\in\mathbb{N}}]_T$ inherits continuity from (finite) cotupling $[-,-]_T$.
- 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 **Cppo**-enrichment of $\mathbf{Set}_T$.
- Continuity of algebraic operations on $T$ follows from continuity of composition $\circ_T$ and cotupling $[-,-]_T$.
- There exists an algebraic operation on $T$ that is not strict.

Continuity of algebraic operations can be confirmed using the bijective correspondence, studied in [30], of an $n$-ary algebraic operation $\alpha$ on $T$ and a $\mathbf{Set}_T$-arrow (called *generic effects*) $\beta\colon 1 \to_T \mathbf{n}$. For a family $\{f_i\colon A \to_T B\}_{i=1}^n$ of $\mathbf{Set}_T$-arrows, the $\mathbf{Set}_T$-arrow $\alpha\{f_i\}_{i=1}^n\colon A \to_T B$ can be equivalently given by

$$A \xrightarrow{\beta\otimes A}_T \mathbf{n} \times A \overset{\cong}{\Rightarrow}_T A + \cdots + A \xrightarrow{[f_1,\ldots,f_n]_T}_T B$$

using the corresponding generic effect $\beta$, where $\mathbf{n}$ is the $n$-fold coproduct of 1.

An example of non-strict algebraic operation is a 1-ary algebraic operation $\mathit{raise}_e^{(1)}$ on the exception monad $\mathcal{E}$ (see Example 2.2.2 & 2.2.5). The algebraic operation $\mathit{raise}_e^{(1)}$ always returns $e \in E$, ignoring its argument, while $\bot \in \mathbf{Set}_{\mathcal{E}}(X, Y) = \mathbf{Set}(X, 1 + E + Y)$ always returns $* \in 1$. □

The domain-theoretic properties of our component calculus support characterization of the Girard style fixed point operator $\mathrm{Fix}_G$ by a limit of finite approximations.

**Definition 3.3.5** (finite approximation $\mathrm{Fix}_G^{(i)}$)**.** For a $T$-transducer $(X, c, x)\colon A + \mathbb{N} \times A \twoheadrightarrow A + \mathbb{N} \times A$ and each $i \in \omega$, the $T$-transducer $\mathrm{Fix}_G^{(i)}(X, c, x)\colon A \twoheadrightarrow A$ is defined to be

$$\mathrm{Tr}^N_{A,A}(\boxplus_{n\in\mathbb{N}}\{F^n(X, c_n^{(i)}, x)\} \circ J((\mathrm{id}_A + \coprod_{i\in\mathbb{N}} \sigma'_{N_i,N_i})^*))$$

where $N$ and $\sigma'$ are defined as in Definition 3.3.2, and $c_n^{(i)}$ is the $\mathbf{Set}_T$-arrow defined by

$$c_n^{(i)} := \begin{cases} \bot & \text{if } i \le n \\ c & \text{otherwise} \end{cases}$$

for each $i \in \omega$ and $n \in \mathbb{N}$.

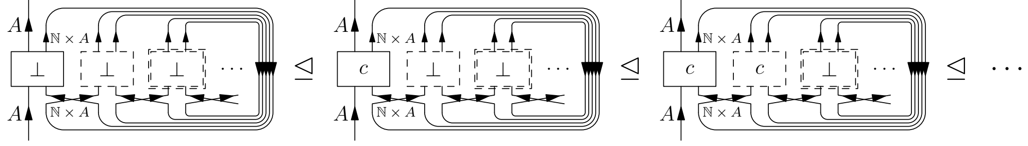



Figure 3.5: The $\omega$-chain of Finite Approximations $\mathrm{Fix}_G^{(i)}(X,c,x)$

**Proposition 3.3.6.** Let $(X,c,x)\colon A+\mathbb{N}\times A \rightarrow A+\mathbb{N}\times A$ be a $T$-transducer. The family $\{\mathrm{Fix}_G^{(i)}(X,c,x)\colon A \rightarrow A\}_{i\in\omega}$ of $T$-transducers forms an $\omega$-chain

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

as depicted in Figure 3.5, and the $T$-transducer $\mathrm{Fix}_G(X,c,x)\colon A \rightarrow A$ satisfies the following behavioral equivalence

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

*Proof.* Since the family $\{(X,c_n^{(i)},x)\colon A+\mathbb{N}\times A \rightarrow A+\mathbb{N}\times A\}_{i\in\omega}$ forms an $\omega$-chain

$$(X,c_n^{(0)},x) \;\trianglelefteq\; (X,c_n^{(1)},x) \;\trianglelefteq\; \cdots$$

and its supremum $\sup_{i\in\omega}(X,c_n^{(i)},x)$ is given by $(X,c,x)$, the statement is an immediate consequence of Lemma 3.3.4, namely continuity of the operators $\circ$, $F$, Tr and $\boxplus_{n\in\mathbb{N}}$. ☐

### 3.3.4 Mackie Style Fixed Point Operator $\mathrm{Fix}_M$

Girard interprets recursion in GoI as a limit of finite approximations in [12], and the Girard style fixed point operator $\mathrm{Fix}_G$ follows his approach in the sense of Proposition 3.3.6. This approach to interpret recursion in GoI is in fact not unique. There is another one by Mackie [24], in which recursion is interpreted as proof nets with a single "box" and feedback loops. In our setting Mackie's approach corresponds to translate recursion with a single use of the countable copy operator $F$ and the trace operator Tr, and especially without any use of countable parallel composition $\boxplus_{n\in\mathbb{N}}$. We introduce another fixed point operator $\mathrm{Fix}_M$ on $T$-transducers, that follows Mackie's approach as depicted in Figure 3.3 (c), and therefore call it "Mackie style" fixed point operator.

**Definition 3.3.7** (Mackie style fixed point operator $\mathrm{Fix}_M$)**.** For a $T$-transducer $(X,c,x)\colon A+\mathbb{N}\times A \rightarrow A+\mathbb{N}\times A$, the $T$-transducer $\mathrm{Fix}_M(X,c,x)\colon A \rightarrow A$ is defined to be

$$\begin{aligned}\mathrm{Tr}_{A,A}^{\mathbb{N}\times A+\mathbb{N}\times A}(\,&(J(\tilde{\mathrm{e}}'^{*}_{A}) \boxplus J(\mathrm{id}^{*}_{\mathbb{N}\times A+\mathbb{N}\times A})) \circ (J(\tilde{\mathrm{c}}'^{*}_{A}) \boxplus J(\tilde{\mathrm{d}}^{*}_{A}))\\ &\circ F(X,c,x)\\ &\circ (J(\tilde{\mathrm{c}}^{*}_{A}) \boxplus J(\tilde{\mathrm{d}}'^{*}_{A})) \circ (J(\tilde{\mathrm{e}}^{*}_{A}) \boxplus J(\sigma'^{*}_{\mathbb{N}\times A,\mathbb{N}\times A}))\,)\end{aligned}$$

where $\sigma'$ is swapping defined as in Definition 3.3.2.

This Mackie style fixed point operator $\mathrm{Fix}_M$ coincides with the Girard style fixed point operator $\mathrm{Fix}_G$, therefore both two styles of fixed point operators enjoy useful properties shown in Proposition 3.3.3 & 3.3.6.

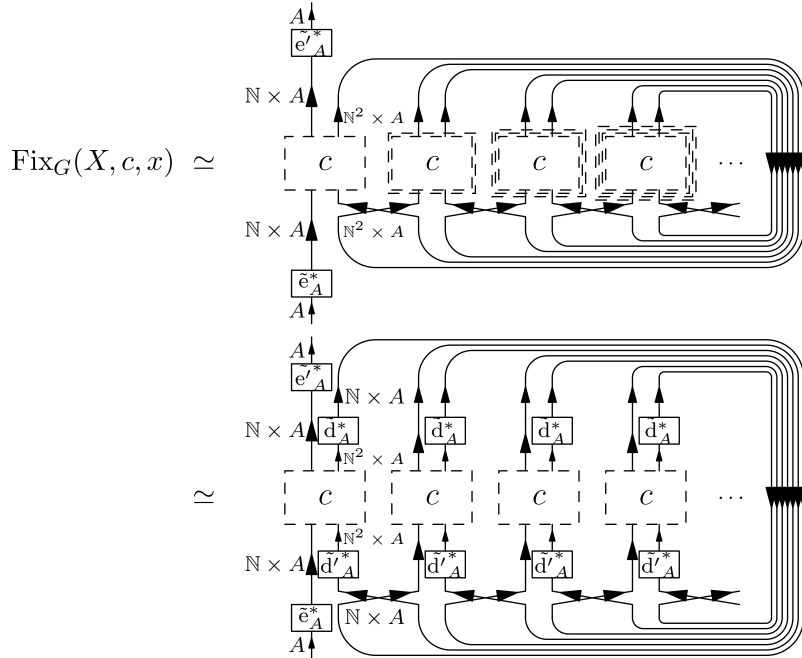
**Theorem 3.3.8** (coincidence of two styles of fixed point operator)**.** The following behavioral equivalence

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

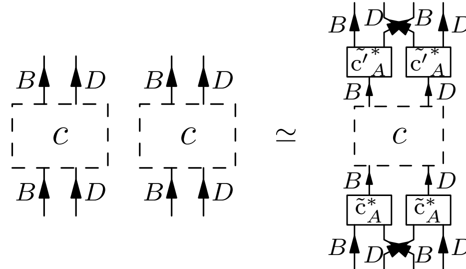
holds for any $T$-transducer $(X, c, x)\colon A + \mathbb{N} \times A \rightharpoonup A + \mathbb{N} \times A$.

*Proof.* We use diagrammatic depictions to illustrate the proof, in which naturality of the trace operator Tr is often used implicitly. What is crucial in the proof is *monoidal* naturality of primitive $T$-transducers $J(\tilde{\mathrm{d}}^*), J(\tilde{\mathrm{d}'}^*), J(\tilde{\mathrm{c}}^*), J(\tilde{\mathrm{c}'}^*)$. It holds because the underlying **Set**-arrows $\tilde{d}$ and $\tilde{c}$ are monoidal natural with respect to coproducts $(+, \emptyset)$ of **Set**, and the pairs $\tilde{\mathrm{d}} \cong \tilde{\mathrm{d}}'$ and $\tilde{\mathrm{c}} \cong \tilde{\mathrm{c}'}$ are in fact isomorphic.

The first step is to "flatten" nested use of the countable copy operator $F$ in the operator $\mathrm{Fix}_G$, exploiting naturality of the primitive $T$-transducer $J(\tilde{\mathrm{d}}^*)$. For the technical reason we add an extra nest of the operator $F$ before flattening them, using naturality of the primitive $T$-transducer $J(\tilde{\mathrm{e}}^*)$ and the fact that the operator $F$ yields a traced symmetric monoidal functor on the category $\mathbf{Res}(T)$.

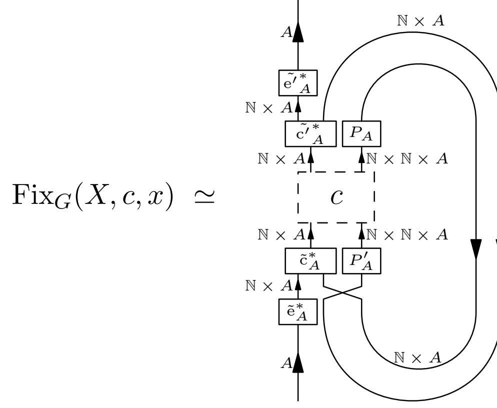



The second step is to get rid of countable parallel composition $\boxplus_{n\in\mathbb{N}}$. Two $T$-transducers in the form of $F(X, c, x)\colon B + D \rightharpoonup B + D$ can be "joined" via monoidal naturality of primitive $T$-transducers $J(\tilde{\mathrm{c}}), J(\tilde{\mathrm{c}})$ as below:

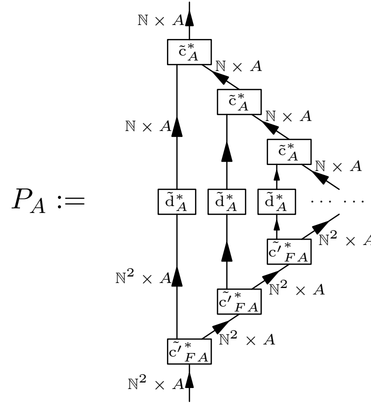



and in this way the countable number of $T$-transducers in the form $F(X, c, x)$

can be "joined."

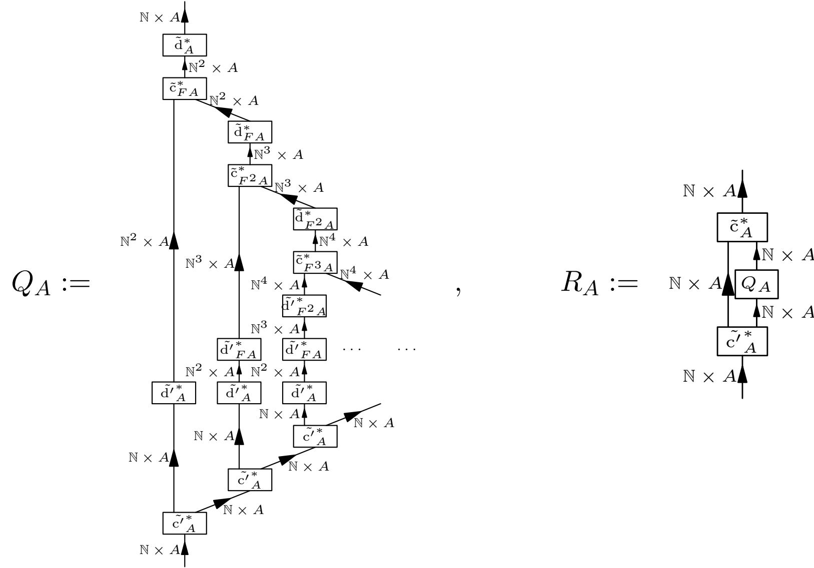



We define the $T$-transducer $P_A\colon \mathbb{N} \times \mathbb{N} \times A \rightarrowtriangle \mathbb{N} \times A$ by




and obtain the $T$-transducer $P'_A$ from $P_A$ by replacing $J(\tilde{d})$ with $J(\tilde{d}')$, $J(\tilde{d})$ with $J(\tilde{c}')$ and vice versa.

The last step is rather tricky. Let $Q_A\colon \mathbb{N}\times A \rightarrowtriangle \mathbb{N}\times A$ and $R_A\colon \mathbb{N}\times A \rightarrowtriangle \mathbb{N}\times A$ be $T$-transducers defined as below.




We can obtain $T$-transducers $Q'_A, R'_A$ from $Q_A, R_A$ in the same way as we obtain $P'_A$ from $P_A$. Since $J(\tilde{d}')$ and $J(\tilde{c}')$ are inverses of $J(\tilde{d})$ and $J(\tilde{c})$ respectively, $P'_A, Q'_A, R'_A$ yield inverses of $P_A, Q_A, R_A$ respectively. Additionally $R_A$ and $R'_A$

satisfies monoidal naturality since $J(\tilde{\mathrm{d}}), J(\tilde{\mathrm{d}'}), J(\tilde{\mathrm{d}}), J(\tilde{\mathrm{c}'})$ are monoidal natural. Therefore by introducing $T$-transducers $Q_A, Q'_A$ using the behavioral equivalence

$$Q_A \circ P_A \simeq \tilde{\mathrm{d}}^*_A \circ \tilde{\mathrm{c}}^*_{FA} \circ (\mathbb{N}\times\mathbb{N}\times A \otimes Q_{FA}) \circ \tilde{\mathrm{c}'}^*_{FA}$$

we obtain the desired behavioral equivalence.

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

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

$\square$

# Chapter 4

# Adequate Translation of Effectful Terms to Transducers

## 4.1 Translation to Transducers

We give a concrete translation $(\!|-|\!)$ of terms in a language $\mathcal{L}_\Sigma$ to $T$-transducers, choosing a monad $T$ that satisfies Requirement 2.2.1 and "supports" the algebraic signature $\Sigma$ in the sense of Definition 2.2.6. The definition of the translation $(\!|-|\!)$ is by means of our component calculus over $T$-transducers, and given using diagrammatic depictions. Our translation $(\!|-|\!)$ is precisely an extension of the translation given by the mGoI framework to recursion.

**Definition 4.1.1** (translation $(\!|-|\!)$)**.** Let $\Gamma$ be a finite list $\mathtt{x}_1 : \tau_1, \dots \mathtt{x}_m : \tau_m$ and $\Gamma \vdash \mathtt{M} : \tau$ be a type judgement. The $T$-transducer

$$(\!|\Gamma \vdash \mathtt{M} : \tau|\!) = \boxed{(\!|\Gamma \vdash \mathtt{M} : \tau|\!)} \;:\; \coprod_{i=0}^{m} \mathbb{N} \rightharpoonup \coprod_{i=0}^{m} \mathbb{N}$$

(the box has $\mathbb{N}$-labelled edges: one plus $m$ on top and one plus $m$ on the bottom)

is inductively defined as in Figure 4.1–4.3, where we omit labels of edges (either $\mathbb{N}$ or $\mathbb{N} \times \mathbb{N}$) and write e.g. c instead of c$^*$, for visibility.

In translation of recursion depicted in Figure 4.2 we implicitly use the Mackie style fixed point operator $\mathrm{Fix}_M$. The Girard style fixed point operator $\mathrm{Fix}_G$ can be used instead as depicted in Figure 4.4, due to Theorem 3.3.8.

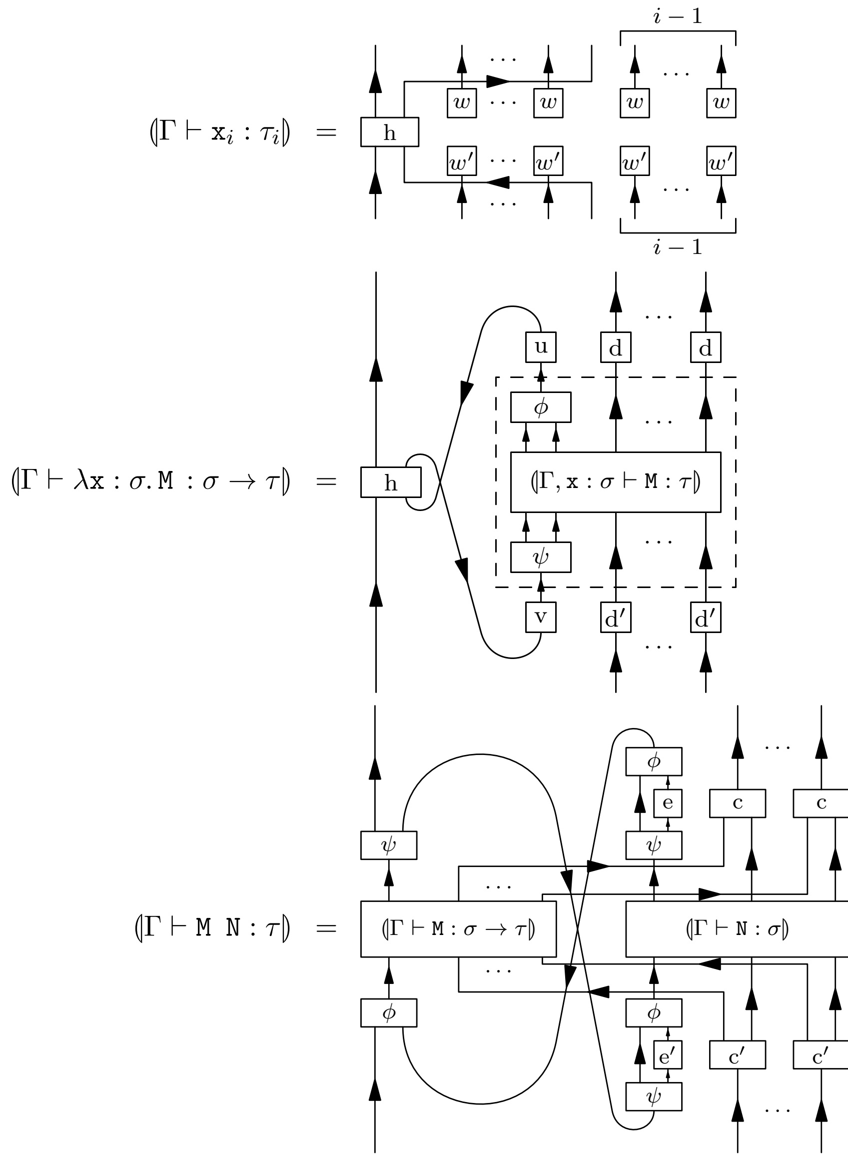



Figure 4.1: Inductive Definition of the Translation (|−|): Core Fragments

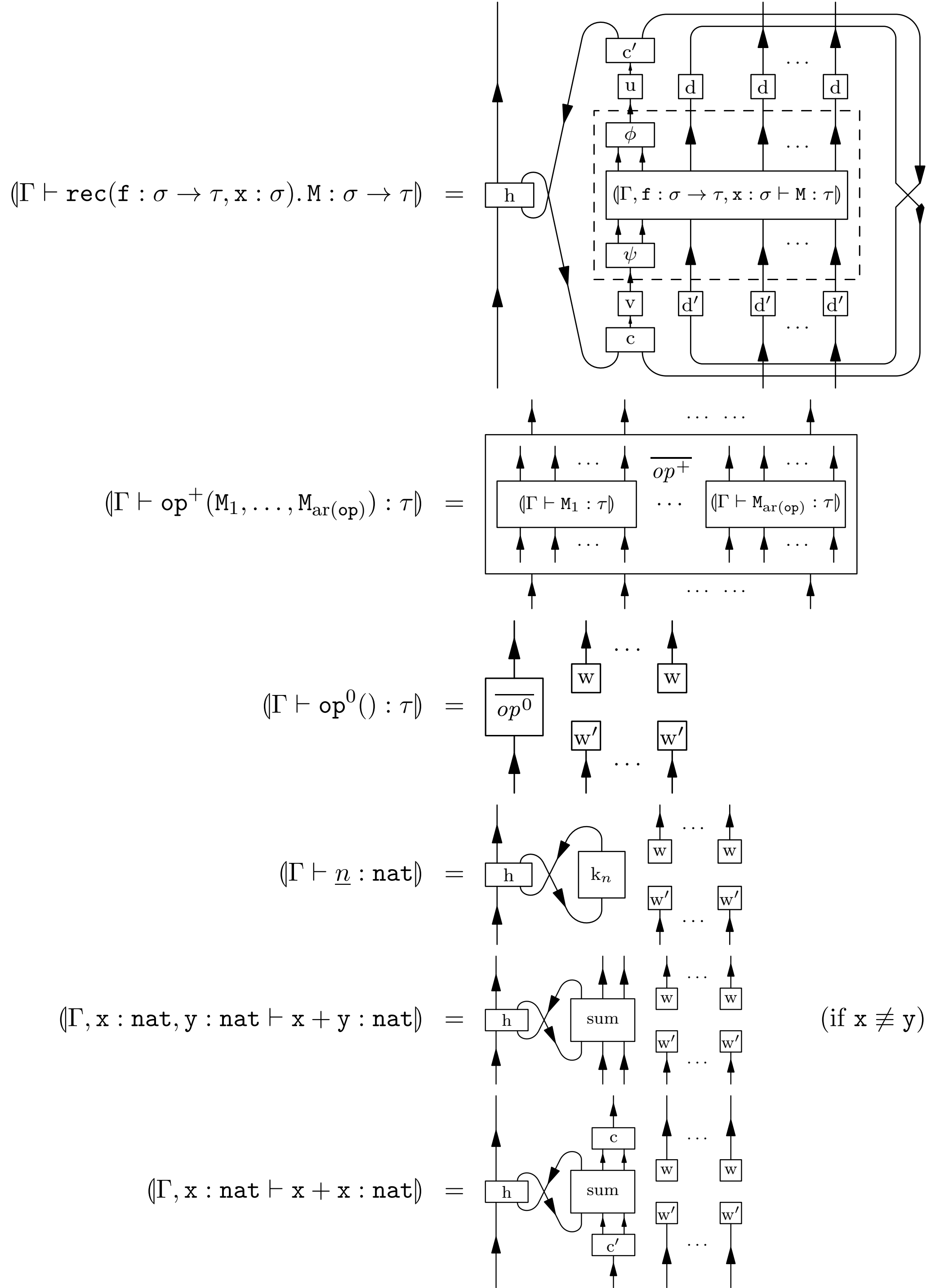



Figure 4.2: Inductive Definition of the Translation (|−|): Recursion, Operations in Σ and Arithmetic Primitives

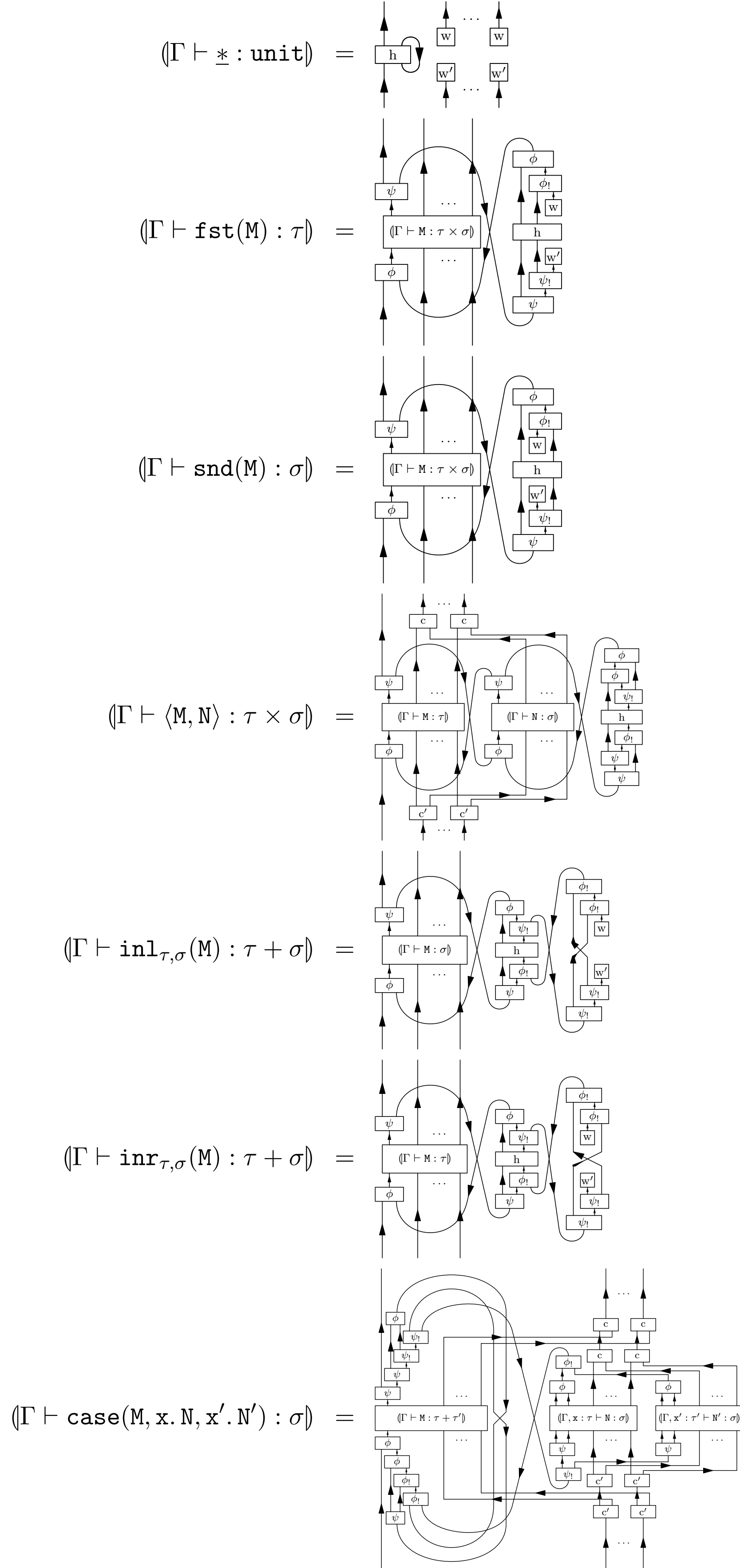



Figure 4.3: Inductive Definition of the Translation $(\!|-|\!)$: Product Types and Coproduct Types

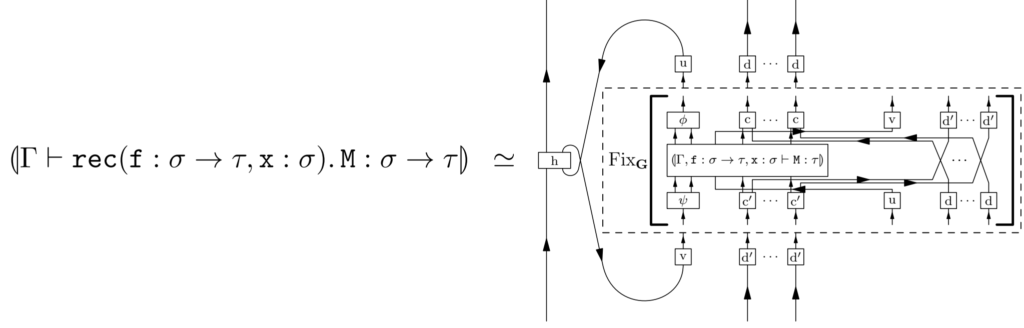



Figure 4.4: Equivalent Translation of Recursion by $\mathrm{Fix}_G$

### 4.1.1 Underlying Categorical Model

The original translation in the mGoI framework is extracted from a categorical interpretation on the model $\mathbf{Per}_\Phi$. The model $\mathbf{Per}_\Phi$, developed in [18, Section 6], is the Kleisli category of a monad $\Phi\colon \mathbf{Per} \to \mathbf{Per}$. Similarly our translation $(\!|-|\!)$ is backed up by a modified categorical model $\mathbf{Per}_{\Phi'}$, that is the Kleisli category of a modified monad $\Phi'\colon \mathbf{Per} \to \mathbf{Per}$. We briefly describe how our model $\mathbf{Per}_{\Phi'}$ is constructed, without going into technical details.

The category $\mathbf{Per}$ is a bicartesian category obtained via categorical GoI combined with the realizability construction (see e.g. [22]). Since the category $\mathbf{Res}(T)$ of resumptions (i.e. equivalence classes of $T$-transducers modulo behavioral equivalence $\simeq$) forms a GoI situation, categorical GoI provides an SK-algebra $(\mathbf{Res}(T)(\mathbb{N},\mathbb{N}),\cdot)$ as stated in Proposition 3.1.5. The realizability construction on the SK-algebra $(\mathbf{Res}(T)(\mathbb{N},\mathbb{N}),\cdot)$ yields the bicartesian closed category $\mathbf{Per}$ of *partial equivalence relations* on $\mathbf{Res}(T)(\mathbb{N},\mathbb{N})$.

The model $\mathbf{Per}_{\Phi'}$ is obtained as the Kleisli category of the strong monad $\Phi'$ on $\mathbf{Per}$. The monad $\Phi'$ is induced by an adjunction

$$\mathbf{CAdm}_\perp \underset{\hookrightarrow}{\overset{\longleftarrow}{\perp}} \mathbf{Per}$$

between $\mathbf{Per}$ and its (non-full) subcategory $\mathbf{CAdm}_\perp$. The subcategory $\mathbf{CAdm}_\perp$ is defined by picking up "well-behaved" objects and arrows in $\mathbf{Per}$. Intuitively objects and arrows in $\mathbf{CAdm}_\perp$ respect lifted algebraic operations $\overline{\alpha}$ on $T$-transducers (and hence resumptions) and the $\omega$-cpo structure $(\mathbf{Trans}(T)(\mathbb{N},\mathbb{N}),\trianglelefteq)$ on $T$-transducers with minimal elements in the form of $(X,\perp,x)$ (see Section 3.3.3).

Note that the monad $\Phi'$ is a modification of the original monad $\Phi$ used in the mGoI framework. If we relax the notion of "well-behavior" and use another subcategory $\mathbf{CPer}$ of $\mathbf{Per}$, whose objects and arrows respect lifted algebraic operations $\overline{\alpha}$ only, we have another adjunction

$$\mathbf{CPer} \underset{\hookrightarrow}{\overset{\longleftarrow}{\perp}} \mathbf{Per}$$

and it induces the monad $\Phi$.

Because the category $\mathbf{Per}$ is bicartesian closed and $\Phi'$ is a strong monad on it, the Kleisli category $\mathbf{Per}_{\Phi'}$ gives a categorical model of the language $\mathcal{L}_\Sigma$ as in [24, 29]. Our concrete translation $(\!|-|\!)$ is extracted from a categorical interpretation

of $\mathcal{L}_\Sigma$ on the model $\mathbf{Per}_{\Phi'}$. In particular we can prove that algebraic operations on $T$ injectively induces algebraic operations on $\Phi'$, like in [18, Theorem 6.1], and that the Girard style fixed point operator $\mathrm{Fix}_G$ yields a (categorical) fixed point operator of $\mathbf{Per}_{\Phi'}$. These results justify our translation of operations $\mathtt{op} \in \Sigma$ by lifted algebraic operations $\overline{op}$ on $T$-transducers, and of recursion by the Girard style fixed point operator $\mathrm{Fix}_G$ (and equivalently the Mackie style fixed point operator $\mathrm{Fix}_M$).

### 4.2 Adequacy of Translation

Our adequacy result relates the translation $(\!|-|\!)$, extracted from categorical (denotational) semantics, to operational semantics described in Section 2.1.2. The statement of adequacy is in Plotkin and Power's sense [29], in which evaluation results of terms are "collected" via both $T$-transducers and effect values.

For a closed term $\mathtt{M}$ of base type $\mathtt{nat}$, we can observe that executing the $T$-transducer $(\!|\mathtt{M}|\!) \colon \mathbb{N} \multimap \mathbb{N}$ with input in the form of $\mathtt{dd}\langle i, m\rangle$ produces output in the form of $\mathtt{dd}\langle i, n\rangle$, where $i, m \in \mathbb{N}$ can be arbitrary and $n \in \mathbb{N}$ corresponds to a possible evaluation result of the term $\mathtt{M}$ (see Section 3.2.2 for the notation). On this observation we fix the retraction $\mathrm{enc} : \mathbb{N} \lhd \mathbb{N} : \mathrm{dec}$ such that

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

and "collect" execution results of a $T$-transducer $(X, c, x) \colon \mathbb{N} \multimap \mathbb{N}$ by taking

$$(X, c, x)^\dagger := ((\pi'_{X,\mathbb{N}})^* \circ_T c)(x, \mathrm{enc}(m_0)) \in T\mathbb{N}$$

where $\pi'_{X,\mathbb{N}} \colon X \times \mathbb{N} \to \mathbb{N}$ is the second projection of $\mathbf{Set}$ and $m_0$ is a fixed natural number. This procedure $(-)^\dagger$ executes a given $T$-transducer from its initial state with input data $\mathrm{enc}(m)$, and gathers output data, ignoring internal states that memorizes history of effect occurrences in execution. Two behavioral equivalent $T$-transducers $(X, c, x) \simeq (Y, d, y) : \mathbb{N} \multimap \mathbb{N}$ indeed produce the same "collection" of execution results, i.e. $(X, c, x)^\dagger = (Y, d, y)^\dagger$ holds.

To "collect" evaluation results of terms via effect values, we utilize the fact that the set $T\mathbb{N}$, isomorphic to the set $\mathbf{Set}_T(1, \mathbb{N})$, is a continuous $\Sigma$-algebra. Since the monad $T$ satisfies Requirement 2.2.1 and *supports* the algebraic signature $\Sigma$ in the sense of Definition 2.2.6, each operation $\mathtt{op} \in \Sigma$ comes with an $\mathrm{ar}(\mathtt{op})$-ary algebraic operation $op$ on $T$ and it gives an $\mathrm{ar}(\mathtt{op})$-ary continuous function on the $\omega$-cppo $\mathbf{Set}_T(1, \mathbb{N})$.

Let $\mathbf{CVal}_{\mathtt{nat}}$ be the set of closed values of type $\mathtt{nat}$. For a closed term $\mathtt{M}$ of base type $\mathtt{nat}$, its effect value $|\mathtt{M}|$ is an element of the free continuous $\Sigma$-algebra $CT_\Sigma(\mathbf{CVal}_{\mathtt{nat}})$ on the set $\mathbf{CVal}_{\mathtt{nat}}$ (see Section 2.1.2). Therefore we can obtain a strict homomorphism $[\![-]\!] \colon CT_\Sigma(\mathbf{CVal}_{\mathtt{nat}}) \to T\mathbb{N}$ between continuous $\Sigma$-algebras by lifting the function

$$\mathbf{CVal}_{\mathtt{nat}} \overset{\cong}{\Rightarrow} \mathbb{N} \xrightarrow{\mathrm{enc}^*} T\mathbb{N}\ .$$

Note that all elements of the set $\mathbf{CVal}_{\mathtt{nat}}$ is in the form of $\underline{n}$ and we can take the canonical isomorphism $\mathbf{CVal}_{\mathtt{nat}} \overset{\cong}{\Rightarrow} \mathbb{N}$ in $\mathbf{Set}$ that assigns $\underline{n}$ to $n$.

Given a closed term $\mathtt{M}$ of base type $\mathtt{nat}$, we "collect" its evaluation results via the effect value $|\mathtt{M}|$ by taking $[\![|\mathtt{M}|]\!] \in T\mathbb{N}$. Recall that the effect value $|\mathtt{M}|$ can be understood as a (possibly infinite) tree whose leaves correspond to possible evaluation results of $\mathtt{M}$. The procedure $[\![-]\!]$ gathers all leaves of $|\mathtt{M}|$, forgetting the branching structure of $|\mathtt{M}|$ that represents history of effect occurrences in

evaluation of $\mathtt{M}$. For example, if we take $\Sigma = \Sigma_{\mathrm{nondet}}$ from Example 2.1.1 and $T = \mathcal{P}$ from Example 2.2.2, effect values of two terms $\mathtt{choose}(\underline{0}, \mathtt{choose}(\underline{1}, \underline{2}))$ and $\mathtt{choose}(\mathtt{choose}(\underline{0}, \underline{1}), \underline{2})$ are distinct but identified as $\{\mathrm{enc}(0), \mathrm{enc}(1), \mathrm{enc}(2)\} \in \mathcal{P}\mathbb{N}$ via the procedure $[\![-]\!]$.

With two procedures $(-)^{\dagger}$ and $[\![-]\!]$ in hand, adequacy of our translation $(\!|-|\!)$ is precisely stated as below.

**Theorem 4.2.1** (adequacy of $(\!|-|\!)$)**.** Any closed term $\mathtt{M}$ of base type $\mathtt{nat}$ satisfies $(\!|\mathtt{M}|\!)^{\dagger} = [\![|\mathtt{M}|]\!]$.

### 4.2.1 Proof of Adequacy

To prove Theorem 4.2.1 we introduce a language $\overline{\mathcal{L}_{\Sigma}}$ following [29]. It is made from the target language $\mathcal{L}_{\Sigma}$ by replacing the term constructor $\mathtt{rec}$ with $\mathtt{rec}_n$ for each $n \in \mathbb{N}$ and adding the non-value constant $\Omega_{\tau}$ for each type $\tau$. Transition relation $\to$ for a redex $\mathtt{rec}_n(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M}$ is defined by

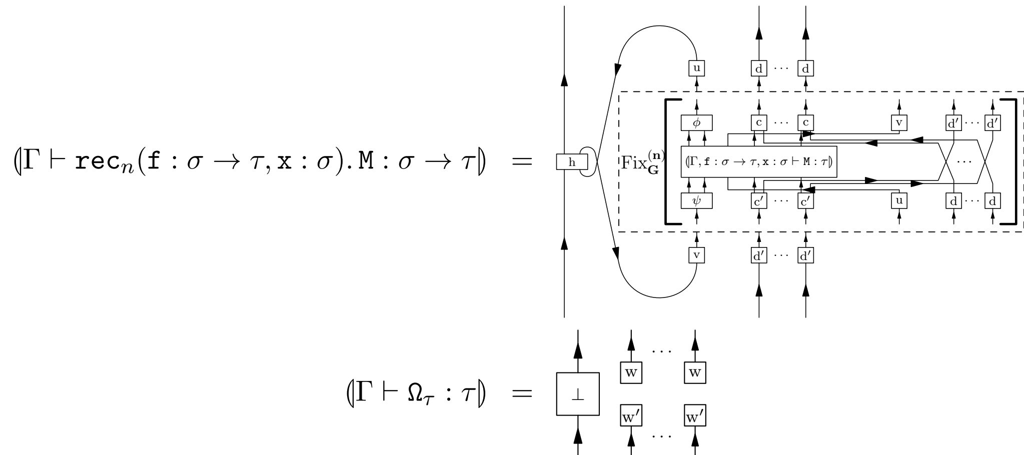
$$\begin{aligned} &\mathtt{rec}_{n+1}(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M} \to (\lambda \mathtt{x} : \sigma.\, \mathtt{M})[\mathtt{rec}_n(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M}/\mathtt{f}] \\ &\mathtt{rec}_0(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M} \to \lambda \mathtt{x} : \sigma.\, \Omega_{\tau} \ . \end{aligned}$$

In the way analogous to $\mathcal{L}_{\Sigma}$, any closed term $\mathtt{M}$ of $\overline{\mathcal{L}_{\Sigma}}$ can be uniquely decomposed into one of the forms of $\mathtt{V}$, $\mathtt{E}[\mathtt{R}]$, $\mathtt{E}[\mathtt{op}^0()]$ and $\mathtt{E}[\Omega_{\tau}]$, and its reduction via transition relations $\to, \stackrel{\mathtt{op}_i}{\to}$ is uniquely determined. Additionally the reduction always terminates [29, Lemma 6]. Therefore, in the language $\overline{\mathcal{L}_{\Sigma}}$, the effect value $|\mathtt{M}|$ for a closed term $\mathtt{M}$ of type $\tau$ can be defined by

$$|\mathtt{M}| := \begin{cases} \mathtt{V} & (\mathtt{M} \equiv \mathtt{V}) \\ \mathtt{op}^{+}(|\mathtt{N}_1|, \ldots, |\mathtt{N}_{\mathrm{ar}(\mathtt{op})}|) & (\mathtt{M} \stackrel{\mathtt{op}_i}{\to} \mathtt{N}_i \text{ for each } i = 1, \ldots, \mathrm{ar}(\mathtt{op})) \\ \mathtt{op}^{0}() & (\mathtt{M} \downarrow_{\mathtt{op}}) \\ \Omega & (\mathtt{M} \equiv \mathtt{E}[\Omega_{\tau'}]) \end{cases}$$

It is an element of the free continuous $\Sigma$-algebra $CT_{\Sigma}(\mathbf{CVal}_{\tau})$ over the set $\mathbf{CVal}_{\tau}$ of closed values of type $\tau$. Since the set $\mathbf{CVal}_{\mathtt{nat}}$ includes $\Omega_{\mathtt{nat}}$ other than $\underline{n}$, we define the procedure $[\![-]\!]$: $CT_{\Sigma}(\mathbf{CVal}_{\mathtt{nat}}) \to T\mathbb{N}$ by lifting the function $\mathbf{CVal}_{\mathtt{nat}} \stackrel{\cong}{\to} \mathbb{N} + \{\Omega_{\mathtt{nat}}\} \xrightarrow{[\mathrm{id}^*, \bot]} T\mathbb{N}$ using the least element $\bot$ of the $\omega$-cppo $\mathbf{Set}_T(\{\Omega_{\mathtt{nat}}\}, \mathbb{N})$.

The translation $(\!|\Gamma \vdash \mathtt{rec}_n(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{M} : \sigma \to \tau|\!)$ and $(\!|\Gamma \vdash \Omega_{\tau} : \tau|\!)$ are defined by



where $\perp$ is the least element of the $\omega$-cppo $\mathbf{Set}_T(\mathbb{N}, \mathbb{N})$. We use the $n$-th approximation $\mathrm{Fix}_G^{(n)}$ of the Girard style fixed point operator as in Figure 4.4.

For a closed term $\mathtt{M}$ of $\mathcal{L}_\Sigma$ of base type $\mathtt{nat}$ and for each $n \in \mathbb{N}$, let $\mathtt{M}^{(n)}$ be a term of $\overline{\mathcal{L}_\Sigma}$ made from $\mathtt{M}$ by replacing all occurrences of $\mathtt{rec}$ with $\mathtt{rec}_n$. Theorem 4.2.1 can be proved by:

$$\begin{aligned}
[\![|\mathtt{M}|]\!] &= [\![\sup_{n\in\omega} |\mathtt{M}^{(n)}|]\!] \\
&= \sup_{n\in\omega} [\![|\mathtt{M}^{(n)}|]\!] && \text{(by continuity of } [\![-]\!]) \\
&= \sup_{n\in\omega} (\!|\mathtt{M}^{(n)}|\!)^\dagger \\
&= (\sup_{n\in\omega} (\!|\mathtt{M}^{(n)}|\!))^\dagger && \text{(by continuity of } \circ_T) \\
&= (\!|\mathtt{M}|\!)^\dagger && \text{(by Proposition 3.3.6)}
\end{aligned}$$

where the first equation can be proved as in [29] and the third one is a consequence of the following lemma.

**Lemma 4.2.2** (adequacy of $(\!|-|\!)$ for $\overline{\mathcal{L}_\Sigma}$)**.** Any closed term $\mathtt{M}$ of $\overline{\mathcal{L}_\Sigma}$ of base type $\mathtt{nat}$ satisfies $(\!|\mathtt{M}|\!)^\dagger = [\![|\mathtt{M}|]\!]$.

*Proof.* The proof utilizes logical relations between $T$-transducers and closed terms. Let $\mathbf{CVal}_\tau^+$ be the set obtained by adding closed terms in the form of $\mathtt{rec}_n(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\mathtt{M}$ to the set $\mathbf{CVal}_\tau$, and $\mathbf{CTerm}_\tau$ be the set of closed terms of type $\tau$. For each type $\tau$, a binary relation $R_\tau \subseteq \mathbf{Trans}(T)(\mathbb{N}, \mathbb{N}) \times \mathbf{CVal}_\tau^+$ is inductively defined by

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

where $\kappa_n \colon \mathbb{N} \to \mathbb{N}$ is the constant function to $n$. For a list $\Gamma = \mathtt{x}_1 : \tau_1, \ldots \mathtt{x}_m : \tau_m$, the binary relation $R_\Gamma$ is defined by $R_{\tau_1} \times \cdots \times R_{\tau_m}$. Each $R_\tau \subseteq \mathbf{Trans}(T)(\mathbb{N}, \mathbb{N}) \times \mathbf{CVal}_\tau^+$ is lifted to the least binary relation $\overline{R_\tau} \subseteq \mathbf{Trans}(T)(\mathbb{N}, \mathbb{N}) \times \mathbf{CTerm}_\tau$ such that

- $(\mathbf{d}, \mathtt{M}) \in \overline{R_\tau}$ if $(\mathbf{c}, \mathtt{M}) \in \overline{R_\tau}$ and $\mathbf{c} \simeq \mathbf{d}$
- $(J(\mathrm{h}^*) \bullet' \mathbf{c}, \mathtt{V}) \in \overline{R_\tau}$ if $(\mathbf{c}, \mathtt{V}) \in R_\tau$
- $(\mathbf{c}, \mathtt{M}) \in \overline{R_\tau}$ if $\mathtt{M} \to \mathtt{N}$ and $(\mathbf{c}, \mathtt{N}) \in \overline{R_\tau}$
- $(\overline{op}\{\mathbf{c}_1, \ldots, \mathbf{c}_{\mathrm{ar}(\mathtt{op})}\}, \mathtt{M}) \in \overline{R_\tau}$ if $\mathtt{M} \xrightarrow{\mathtt{op}_i} \mathtt{N}_i$ and $(\mathbf{c}_i, \mathtt{N}_i) \in \overline{R_\tau}$ for each $i = 1, \ldots, \mathrm{ar}(\mathtt{op})$
- $(\overline{op}\{\}, \mathtt{M}) \in \overline{R_\tau}$ if $\mathtt{M} \downarrow_{\mathtt{op}}$
- $(J(\perp), \mathtt{E}[\Omega_{\tau'}]) \in \overline{R_\tau}$ .

Note that the binary relation $\overline{R_\tau}$ is closed under behavioral equivalence. The operator $\bullet'$ on $T$-transducers is an adaptation of binary application $\bullet$. For two $T$-transducers $(X, c, x)\colon A + C \rightharpoonup B + C$ and $(Y, d, y)\colon C \rightharpoonup C$, the $T$-transducer $(X, c, x) \bullet' (Y, d, y)$ is defined to be

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

It can be easily proved that

$$\overline{R_{\mathtt{nat}}} \subseteq \{(\mathbf{c}, \mathtt{M}) \mid \mathbf{c}^\dagger = [\![|\mathtt{M}|]\!]\}$$

by (structural) induction on $\overline{R_{\mathtt{nat}}}$. Let $\Gamma \vdash \mathtt{M} : \tau$ be a type judgement and $(\!|\mathtt{M}|\!)[\vec{\mathbf{c}}]\colon \mathbb{N} \rightharpoonup \mathbb{N}$ be a $T$-transducer defined by $(\cdots(((\!|\mathtt{M}|\!) \bullet' \mathbf{c}_1) \bullet' \mathbf{c}_2) \cdots \bullet' \mathbf{c}_m)$. To prove the desired statement it suffices to prove

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

by induction on $\mathtt{M}$. In the proof below we exploit axioms that are satisfied by our component calculus and listed in Section 3.2.3.

- If $\mathtt{M} \equiv \mathtt{x}_i$, the behavioral equivalence

  $$(\!|\mathtt{x}_i|\!)[\vec{\mathbf{c}}] \simeq J(\mathrm{h}^*) \bullet' \mathbf{c}_i\ \ ,$$

  together with $(\mathbf{c}_i, \mathtt{V}_i) \in R_{\tau_i}$, implies $((\!|\mathtt{x}_i|\!)[\vec{\mathbf{c}}], \mathtt{x}_i[\vec{\mathtt{V}}/\Gamma]) \in \overline{R_{\tau_i}}$.

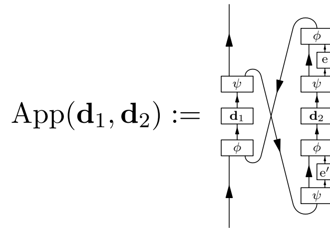
- If $\mathtt{M} \equiv \lambda\mathtt{x} : \sigma.\, \mathtt{N}$, the induction hypothesis implies

  $$(J(\phi^*) \circ (\!|\mathtt{N}|\!)[\vec{\mathbf{c}}] \circ J(\psi^*), \lambda\mathtt{x} : \sigma.\, \mathtt{N}[\vec{\mathtt{V}}/\Gamma]) \in R_{\sigma\to\tau}\ \ .$$

  Therefore it follows that $((\!|\lambda\mathtt{x} : \sigma.\, \mathtt{N}|\!)[\vec{\mathbf{c}}], (\lambda\mathtt{x} : \sigma.\, \mathtt{N})[\vec{\mathtt{V}}/\Gamma]) \in \overline{R_\tau}$ from the behavioral equivalence

  $$(\!|\lambda\mathtt{x} : \sigma.\, \mathtt{N}|\!)[\vec{\mathbf{c}}] \simeq J(\mathrm{h}^*) \bullet' (J(\phi^*) \circ (\!|\mathtt{N}|\!)[\vec{\mathbf{c}}] \circ J(\psi^*))\ \ .$$

- If $\mathtt{M} \equiv \mathtt{N}_1\ \mathtt{N}_2$, let $\mathrm{App}(\mathbf{d}_1, \mathbf{d}_2)\colon \mathbb{N} \rightharpoonup \mathbb{N}$ be a $T$-transducer defined by

  $\mathrm{App}(\mathbf{d}_1, \mathbf{d}_2) :=$ 
  

  for two $T$-transducers $\mathbf{d}_1, \mathbf{d}_2\colon \mathbb{N} \rightharpoonup \mathbb{N}$. Because of the behavioral equivalence

  $$(\!|\mathtt{N}_1\ \mathtt{N}_2|\!)[\vec{\mathbf{c}}] \simeq \mathrm{App}((\!|\mathtt{N}_1|\!)[\vec{\mathbf{c}}], (\!|\mathtt{N}_2|\!)[\vec{\mathbf{c}}])$$

  it suffices to prove that

  $$(\mathbf{d}_1, \mathtt{N}_1) \in \overline{R_{\sigma\to\tau}} \wedge (\mathbf{d}_2, \mathtt{N}_2) \in \overline{R_\sigma}\ \Rightarrow\ (\mathrm{App}(\mathbf{d}_1, \mathbf{d}_2), \mathtt{N}_1\ \mathtt{N}_2) \in \overline{R_\tau}$$

  holds for any $\mathtt{N}_1 \in \mathbf{CTerm}_{\sigma\to\tau}$ and $\mathtt{N}_2 \in \mathbf{CTerm}_\sigma$. The proof can be done by induction on reductions of $\mathtt{N}_1$ and $\mathtt{N}_2$, in which we exploit properties of

lifted algebraic operations $\overline{\alpha}$ (see Section 3.2.3) and strictness of our component calculus (see Lemma 3.3.4). The key observations are the behavioral equivalences

$$\mathrm{App}(\, J(\mathrm{h}^*) \bullet' (J(\phi^*) \circ (\mathbf{e}_1) \circ J(\psi^*)),\, \mathbf{d}_2 \,) \;\simeq$$

$$\mathrm{App}(\, J(\mathrm{h}^*) \bullet' (J(\phi^*) \circ (\mathbf{e}_1) \circ J(\psi^*)),\, J(\mathrm{h}^*) \bullet' \mathbf{e}_2 \,) \;\simeq\; \mathbf{e}_1 \bullet' \mathbf{e}_2$$

that holds for any $T$-transducers $\mathbf{d}_2 \colon \mathbb{N} \multimap \mathbb{N}$, $\mathbf{e}_1 \colon \mathbb{N} + \mathbb{N} \multimap \mathbb{N} + \mathbb{N}$ and $\mathbf{e}_2 \colon \mathbb{N} \multimap \mathbb{N}$.

- If $\mathtt{M} \equiv \mathtt{rec}_n(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{N}$, let $\mathbf{r}_n \colon \mathbb{N} \multimap \mathbb{N}$ be a $T$-transducer defined by

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

Because we have the behavioral equivalence

$$(\!|\mathtt{rec}_n(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{N}|\!)[\vec{\mathbf{c}}] \simeq J(\mathrm{h}^*) \bullet' \mathbf{r}_n$$

it suffices to prove that

$$(\mathbf{r}_n, \mathtt{rec}_n(\mathtt{f} : \sigma \to \tau, \mathtt{x} : \sigma).\, \mathtt{N}[\vec{\mathtt{V}}/\Gamma]) \in R_{\sigma \to \tau}$$

holds for any $n \in \mathbb{N}$. The proof, done by induction on $n$, is supported by the two behavioral equivalences

$$\mathbf{r}_0 \bullet' \mathbf{e} \;\simeq\; J(\bot)$$
$$\mathbf{r}_{n+1} \bullet' \mathbf{e} \;\simeq\; (\!|\mathtt{N}|\!)[\vec{\mathbf{c}}, \mathbf{r}_n, \mathbf{e}]$$

that hold for any $T$-transducer $\mathbf{e} \colon \mathbb{N} \multimap \mathbb{N}$.

- If $\mathtt{M} \equiv \mathtt{op}^+(\mathtt{N}_1, \ldots, \mathtt{N}_{\mathrm{ar}(\mathtt{op})})$, the desired property directly follows from the behavioral equivalence

$$(\!|\mathtt{op}^+(\mathtt{N}_1, \ldots, \mathtt{N}_{\mathrm{ar}(\mathtt{op})})|\!)[\vec{\mathbf{c}}] \simeq \overline{op}((\!|\mathtt{N}_1|\!)[\vec{\mathbf{c}}], \ldots, (\!|\mathtt{N}_{\mathrm{ar}(\mathtt{op})}|\!)[\vec{\mathbf{c}}]) \ .$$

- If $\mathtt{M} \equiv \mathtt{op}^0()$ or $\mathtt{M} \equiv \mathtt{E}[\Omega_{\tau'}]$, the proof is obvious by the definition of $\overline{R_\tau}$.
- The other cases can be proved in the same way as the case $\mathtt{M} \equiv \mathtt{N}_1\ \mathtt{N}_2$.

□

## 4.3 Execution of Resulting Transducers

As shown in Section 2.2.3, the algebraic signature $\Sigma_{\mathrm{prob}}$ for probabilistic choice is supported by the subdistribution monad $\mathcal{D}$, with a binary operation $\mathtt{choose}_p \in \Sigma_{\mathrm{prob}}$ coming with a 2-ary algebraic operation $\oplus_p$ on $\mathcal{D}$. Probabilistic programs, expressed as terms in the language $\mathcal{L}_{\Sigma_{\mathrm{prob}}}$, can therefore be translated to $\mathcal{D}$-transducers in our framework. In this section we use (recursive) probabilistic programs as examples and illustrate execution of the resulting $\mathcal{D}$-transducer.

The first example program is expressed as the following closed term that includes recursive calls of a function.

$$\begin{aligned}&\mathtt{P}_1 \equiv (\mathtt{rec}(\mathtt{flipLoop} : \mathtt{nat} \to \mathtt{nat}, \mathtt{x} : \mathtt{nat}).\, \mathtt{Q})\ \underline{0}\ \ : \mathtt{nat}\\ &\text{where } \mathtt{Q} \equiv \mathtt{choose}_{0.4}(\mathtt{x},\ \mathtt{flipLoop}\,(\mathtt{x} + \underline{1}))\end{aligned}$$

This term $\mathtt{P}_1$ flips an unfair coin repeatedly until head is observed, counting how many tails are observed. Its effect value $|\mathtt{P}_1|$ can be seen as the infinite binary tree shown in Figure 4.5, and the structure of the tree represents how the evaluation of $\mathtt{P}_1$ branches according to the results of coin flipping. Gathering all leaves of the tree yields $[\![|\mathtt{P}_1|]\!] \in \mathcal{D}\mathbb{N}$, that is the (sub)distribution over $\mathbb{N}$ such that $[\![|\mathtt{P}_1|]\!](\mathrm{enc}(n)) = 0.4 \times 0.6^n$ for each $n \in \mathbb{N}$.

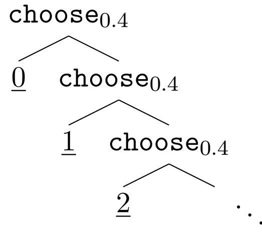



Figure 4.5: The Effect Value $|\mathtt{P}_1|$

The resulting $\mathcal{D}$-transducer $(\!|\mathtt{P}_1|\!) : \mathbb{N} \multimap \mathbb{N}$ can be depicted as in Figure 4.6. Thinking of transducers as token machines, we can visualize execution of the

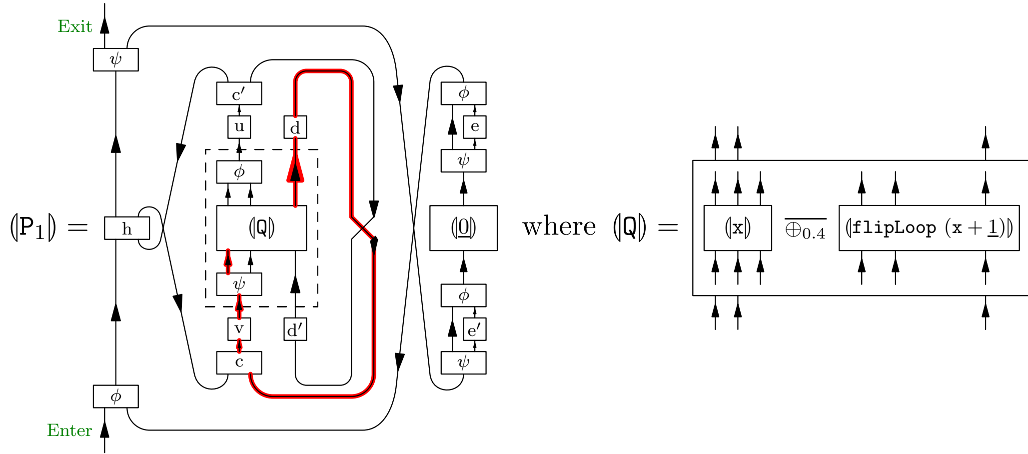


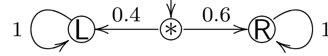

Figure 4.6: The $\mathcal{D}$-transducer $(\!|\mathtt{P}_1|\!) : \mathbb{N} \multimap \mathbb{N}$

$\mathcal{D}$-transducer $(\!|\mathtt{P}_1|\!) : \mathbb{N} \multimap \mathbb{N}$ using a token that carries a natural number as data. A token enters through the open edge labeled with "Enter" carrying the data $\mathrm{enc}(m_0)$, moves around the depiction along edges with updating its data, and hopefully exits though the open edge labeled with "Exit". Theorem 4.2.1 implies that, if a token exits it carries the data $\mathrm{enc}(n)$ with probability $0.4 \times 0.6^n$ for each $n \in \mathbb{N}$.

The way in which a token travels around captures dynamics of evaluation of the term $\mathtt{P}_1$ in some sense. For example we can observe that a token enters a

copy of the "subtransducer" $(\!|\mathtt{Q}|\!)$ as many times as the subterm $\mathtt{Q}$ is evaluated (i.e. the coin is flipped). It in fact enters a different copy of $(\!|\mathtt{Q}|\!)$ for each time, namely it enters the $\mathtt{g0}$-th copy for the first time, the $\mathtt{d}\langle\mathtt{g0},\mathtt{g0}\rangle$-th copy for the second time, the $\mathtt{d}\langle\mathtt{d}\langle\mathtt{g0},\mathtt{g0}\rangle,\mathtt{g0}\rangle$-th copy for the third time and so on. Note that, since the transducer $(\!|\mathtt{Q}|\!)$ is depicted within the dashed box in Figure 4.6, it is "copied" by the operator $F$ and each copy has its own index given by a natural number. Additionally a token goes along the path, indicated by the bold red arrows in Figure 4.6, as many times as the subterm `flipLoop` is recursively called (i.e. the coin results in tail and is flipped again).

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




where labels denote probabilities. The copies of $(\!|\mathtt{Q}|\!)$ memorizes history of probabilistic choices using their internal states. If the coin results in tail twice and in head at the third time, the transducer $(\!|\mathtt{P}_1|\!)$ outputs enc(2) and internal states of the copies of $(\!|\mathtt{Q}|\!)$ result in as below.

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

Readers can see how exactly a token travels around using our tool *TtT*[1]—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 $\mathcal{D}$-transducer by showing how a token moves around and updates its data.

### 4.3.1 Execution Cost

The second example program is expressed as the following closed term.

$$\mathtt{P}_2 \equiv (\lambda\mathtt{x} : \mathtt{nat}.\, \mathtt{x} + \mathtt{x})\ \ \mathtt{choose}_{0.4}(\underline{3} + \underline{4},\ \underline{5} + \underline{6})\ \ :\ \mathtt{nat}$$

It includes no recursive calls of functions and its effect value $|\mathtt{P}_2|$ is equal to the finite binary tree shown in Figure 4.7. We can obtain the distribution $[\![|\mathtt{P}_2|]\!] \in \mathcal{D}\mathbb{N}$ such that $[\![|\mathtt{P}_2|]\!](\mathrm{enc}(14)) = 0.4$ and $[\![|\mathtt{P}_2|]\!](\mathrm{enc}(22)) = 0.6$, and Theorem 4.2.1 implies that the resulting $\mathcal{D}$-transducer $(\!|\mathtt{P}_2|\!) \colon \mathbb{N} \rightharpoonup \mathbb{N}$, roughly depicted in Figure 4.8, outputs enc(14) with probability 0.4 and enc(22) with probability 0.6.

Using this example we focus on execution cost of transducers. Since we adopt the call-by-value evaluation strategy, the subterm $\mathtt{choose}_{0.4}(\underline{3}+\underline{4},\ \underline{5}+\underline{6})$ is evaluated exactly once in evaluation of the term $\mathtt{P}_2$. Therefore *one* probabilistic choice is made by the operation $\mathtt{choose}_{0.4}$ and either $\underline{3}+\underline{4}$ or $\underline{5}+\underline{6}$ is evaluated exactly *once*.

However in execution of the $\mathcal{D}$-transducer $(\!|\mathtt{P}_2|\!)$, we can observe that the "subtransducer" $\overline{\oplus_{0.4}}\{(\!|\underline{3}+\underline{4}|\!), (\!|\underline{5}+\underline{6}|\!)\}$ is executed twice, i.e. a token enters it twice. For the first time a token enters, the subtransducer makes a probabilistic choice, memorizes the result using its internal state, and passes the token to either $(\!|\underline{3}+\underline{4}|\!)$ or $(\!|\underline{5}+\underline{6}|\!)$. When the token comes again, the subtransducer remembers the results

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

$\mathtt{choose}_{0.4}$

$\underline{14}$ $\underline{22}$

Figure 4.7: The Effect Value $|\mathtt{P}_2|$

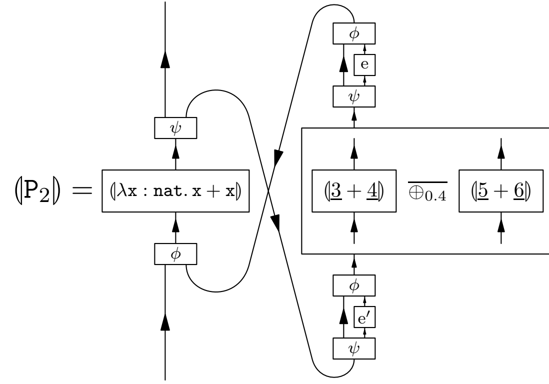



Figure 4.8: The $\mathcal{D}$-transducer $(\!|\mathtt{P}_2|\!) : \mathbb{N} \multimap \mathbb{N}$

of its previous choice using its internal state, and passes the token to the same transducer as the one it passed the token to last time. As a result, in execution of the $\mathcal{D}$-transducer $(\!|\mathtt{P}_2|\!)$, *one* probabilistic choice is made by the operator $\overline{\oplus_{0.4}}$ and either $(\!|\underline{3} + \underline{4}|\!)$ or $(\!|\underline{5} + \underline{6}|\!)$ is executed *twice*.

This unwanted "re-execution" of transducers corresponds to the duplication of the bounded variable $\mathtt{x}$ in the subterm $\lambda\mathtt{x} : \mathtt{nat}.\, \mathtt{x} + \mathtt{x}$. While we force transducers to never "re-generate" effects by exploiting their internal states, transducers are in fact "re-executed" as many times as bounded variables are duplicated. This means that execution cost of transducers is not compatible with evaluation cost of terms, because evaluation cost of terms is not affected by duplication of bounded variables as long as we adopt the call-by-value evaluation strategy.

# Chapter 5

# Conclusion and Future Work

We developed a framework that provides a translation of effectful $\lambda$-terms to transducers, as an extension of the mGoI framework [18]. In the mGoI framework, various algebraic effects are accommodated in a uniform way by employing categorical GoI [1] and categorical semantics of algebraic effects [29], and a translation from effectful $\lambda$-terms to transducers is defined by means of a coalgebraic component calculus over transducers. Our framework inherits all these characteristics of the mGoI framework, and further accommodates recursion that is lacking in the mGoI framework. We defined new two styles of "fixed point" operators on transducers—namely the Girard style and Mackie style fixed point operators—and show their coincidence. They enable us to give a translation from effectful $\lambda$-terms to transducers, that is defined by means of a component calculus over transducers, even for recursion. The translation was proved to be adequate.

We can say our adequacy result is essentially due to the primitive transducer $J(\mathrm{h}^*)$ (see Section 3.2.2). It is well known that program semantics based on GoI has the "call-by-name" nature, in the sense that function application is interpreted as interactions of a function and its arguments where arguments are evaluated as many times as they are called by the function. We utilize the primitive transducer $J(\mathrm{h}^*)$ as a CPS-like construct to model the call-by-value evaluation strategy in token machine semantics. There exists another but similar approach to model the call-by-value evaluation strategy in GoI-based semantics, namely Schöpp's approach in [32] to employ a refinement of Plotkin's call-by-value CPS-transformation [28]. The relationship between our CPS-like construct and Schöpp's refined CPS-transformation is yet to be examined.

Since transducers can be understood as "effectful and memoryful" token machines, we can think of extracting a compilation technique of effectful terms to hardware, from our framework, by implementing transducers on hardware as in [24, 6]. In extracting the compilation technique, 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. They both can inflate state spaces of transducers to infinite ones, and therefore can make hardware implementation of transducers consume much more resources. In our translation of recursion (see Figure 4.2) we reduce occurrences of these two operators, especially exclude occurrences of countable parallel composition $\boxplus_{n\in\mathbb{N}}$, using the Mackie style fixed point operator instead of the Girard style one. To deal with the countable copy operator $F$ that still appears in our translation, bounded linear logic [9] might provide a useful way to statically estimate how many copies will be actually executed and to implement transducers using finite resources.

For compilation techniques it is important not only to consume less resources but also to execute compiled programs faster. This faster execution of compiled

programs corresponds to execution of resulting transducers at less cost in our framework. We observed in Section 4.3.1, however, that execution cost of resulting transducers is unnecessarily high. To be concrete our framework does not respect the cost model of the call-by-value evaluation strategy, in the sense that execution cost of resulting transducers is not compatible with evaluation cost of terms in the call-by-value evaluation strategy.

This cost problem is not specific to our setting and in fact applies to token machine semantics in general. There have been proposed some approaches to make token machine semantics respect the cost model of a specific evaluation strategy, and they would help us deal with the cost problem in our framework. For example Fernández and Mackie propose a dynamic jumping mechanism of token machines in [5] to make their token machine semantics respect the call-by-value evaluation strategy. Their approach is influenced by Danos and Regnier's work [4] that proves token machine semantics can respect the cost model of the call-by-name evaluation strategy if one introduce a (static) jumping mechanism to token machines. Additionally Dal Lago et al. in [3] obtain token machine semantics that respects the call-by-value evaluation strategy, in which token machines are generalized to "multi-token machines" that can process several tokens at once.

# References


[1] Samson Abramsky, Esfandiar Haghverdi, and Philip J. Scott. Geometry of Interaction and linear combinatory algebras. *Math. Struct. in Comp. Sci.*, 12(5):625–665, 2002.

[2] L. S. Barbosa. Towards a calculus of state-based software components. *Journ. of Universal Comp. Sci.*, 9(8):891–909, 2003.

[3] Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. Parallelism and synchronization in an infinitary context. In *LICS 2015*, pages 559–572. IEEE, 2015.

[4] Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal lambda-calculus. *Elect. Notes in Theor. Comp. Sci.*, 3:40–60, 1996.

[5] Maribel Fernández and Ian Mackie. Call-by-value lambda-graph rewriting without rewriting. In *ICGT 2002*, volume 2505 of *LNCS*, pages 75–89. Springer, 2002.

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

[7] Dan R. Ghica and Alex Smith. Geometry of Synthesis II: from games to delay-insensitive circuits. *Elect. Notes in Theor. Comp. Sci.*, 265:301–324, 2010.

[8] Dan R. Ghica and Alex Smith. Geometry of Synthesis III: resource management through type inference. In *POPL 2011*, pages 345–356. ACM, 2011.

[9] Dan R. Ghica and Alex Smith. Bounded linear types in a resource semiring. In *ESOP 2014*, volume 8410 of *Lect. Notes Comp. Sci.*, pages 331–350. Springer, 2014.

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

[11] Jean-Yves Girard. Geometry of Interaction I: interpretation of system F. In *Logic Colloquium 1988*, volume 127 of *Studies in Logic & Found. Math.*, pages 221–260. Elsevier, 1989.

[12] Jean-Yves Girard. Geometry of Interaction II: deadlock-free algorithms. In *COLOG-88*, volume 417, pages 76–93. Springer, 1990.

[13] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In *POPL 1992*, pages 15–26. ACM, 1992.

[14] Masahito Hasegawa. The uniformity principle on traced monoidal categories. *Publ. RIMS*, 40(3):991–1014, 2004.

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

[16] Ichiro Hasuo and Naohiko Hoshino. Semantics of higher-order quantum computation via Geometry of Interaction. In *LICS 2011*, pages 237–246. IEEE Computer Society, 2011.

[17] Ichiro Hasuo and Bart Jacobs. Traces for coalgebraic components. *Math. Struct. in Comp. Sci.*, 21(2):267–320, 2011.

[18] Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful Geometry of Interaction: from coalgebraic components to algebraic effects. In *CSL-LICS 2014*, page 52. ACM, 2014.

[19] Andre Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Math. Proc. Cambridge Phil. Soc.*, 119(3):447–468, 1996.

[20] Anders Kock. Strong functors and monoidal monads. *Arch. Math.*, 23:113–120, 1972.

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

[22] John Longley. *Realizability toposes and language semantics.* PhD thesis, University of Edinburgh, 1994.

[23] Saunders Mac Lane. *Categories for the working mathematician.* Springer, 1998.

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

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

[26] Koko Muroya, Naohiko Hoshino, and Ichiro Hasuo. Memoryful Geometry of Interaction II: recursion and adequacy. In *POPL 2016*, 2016. To appear.

[27] Jorge Sousa 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.

[28] Gordon Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comp. Sci.*, 1(2):125–259, 1975.

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

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

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

[32] Ulrich Schöpp. Call-by-value in a basic logic for interaction. In *APLAS 2014*, volume 8858 of *Lect. Notes Comp. Sci.*, pages 428–448. Springer, 2014.