Categorical Descriptional Composition

Shin-ya Katsumata

Research Institute for Mathematical Sciences
Kyoto University, Kyoto, 606-8502, Japan
sinya@kurims.kyoto-u.ac.jp

Abstract. Descriptional composition is a method to fuse two term transformation algorithms described by attribute couplings (AC, attribute grammars over terms) into one. In this article, we provide a general categorical framework for the descriptional composition based on traced symmetric monoidal categories and the **Int** construction by Joyal et al. We demonstrate that this framework can handle the descriptional composition of SSUR-ACs, nondeterministic SSUR-ACs, quasi-SSUR ACs and quasi-SSUR stack ACs.

1 Introduction

Descriptional composition [6–8] is a method to fuse two term transformation algorithms described by attribute couplings (ACs; attribute grammars over terms) into one. The AC yielded by the descriptional composition computes the composition of two ACs without constructing the intermediate data structure passed between two ACs; hence it saves time and space in many cases. The descriptional composition was first introduced as an optimisation method for compilers described by ACs [7]. Around the same time Bartha introduced a similar composition method for linear attributed tree transformations [1]. Later, it was realised that ACs can be used to represent functional programs with accumulating parameters [10], and the descriptional composition inspired various fusion transformations of such functional programs [19, 22].

The descriptional composition was first given for the ACs consisting only of term constructors [7]. Later, extensions of the ACs have been studied in [18, 20, 3]. In [3], Boyland considered an extension of ACs with conditional expressions. In [18], Nakano introduces stacks to ACs so that complex parsing functions can be expressed. In each work, the descriptional composition was also considered to these extended ACs. In general, the descriptional composition is sensitive on the language describing ACs, and formulating the descriptional composition verifying its correctness tend to be involved when expressive power of the language increases.

The question we address here is to find a mathematical framework that can uniformly treat these extensions of ACs and the descriptional composition of them. In this paper, we propose such a general categorical framework based on the theory of *traced symmetric monoidal categories* (TSMCs) and the **Int** *construction* by Joyal et al. The key observation in the categorical treatment of ACs and the descriptional composition is that every attribute grammar determines a traced symmetric strict monoidal functor

$F : \mathcal{L}(\Sigma) \to \mathbf{Int}(\mathbb{C})$ where $\mathcal{L}(\Sigma)$ is a free TSMC over a signature Σ, and especially every AC satisfying *syntactic single use condition (SSUR)*, which is the essential condition for the descriptional composition to work, determines a traced symmetric monoidal functor $G : \mathcal{L}(\Sigma) \to \mathbf{Int}(\mathcal{L}(\Delta))$. With this functorial presentation, the descriptional composition becomes the composition of functors, and its associativity can be easily verified. This story scales up to the TSMCs with extra structures, provided that the **Int** construction is also extended to such TSMCs; examples include nondeterministic ACs, quasi-SSUR ACs (an affine version of SSUR AC), and ACs with stacks [18].

In this paper we do not exclude circular terms and recursive computation as meaningless things. Hence every AG assigns some meaning to a given term.

Conventions and Notations Signatures are all many-typed and first-order. We reserve Δ, Σ, Ξ for ranging over signatures. By $\rho \in \Sigma$ and $o \in \Sigma^{\rho_1 \cdots \rho_n \to \rho}$ we mean that ρ is a type of Σ and o is an operator of type $\rho_1 \cdots \rho_n \to \rho$, respectively. We declare the signature for binary trees, cons-lists and natural numbers by

$$\Sigma_{\text{tree}} = (\{*\}, \mathsf{L}^*, \mathsf{N}^{**\to *}), \quad \Sigma_{\text{list}} = (\{*\}, []^*, a :: (-)^{*\to *}), \quad \Sigma_{\text{nat}} = (\{*\}, \mathsf{Z}^*, \mathsf{S}^{*\to *}).$$

For a type $\sigma \in \Sigma$ and sequence of Σ-types $\rho_1 \cdots \rho_n$, by $T_\Sigma^\sigma(\rho_1 \cdots \rho_n)$ we mean the set of open Σ-terms that may contain some variables x_i of type ρ_i $(1 \le i \le n)$. We then extend this notation by $T_\Sigma^{\sigma_1 \cdots \sigma_m}(\rho_1 \cdots \rho_n) = T_\Sigma^{\sigma_1}(\rho_1 \cdots \rho_n) \times \cdots \times T_\Sigma^{\sigma_m}(\rho_1 \cdots \rho_n)$. By Σ^+ we mean that Σ contains a special type #; such signature is called *rooted*. We assume that the readers are familiar with the concept of symmetric monoidal categories [17]. We also employ the 2-category theory and the theory of pseudo-monads and pseudo-distributive laws; see e.g. [21].

2 Classical Attribute Couplings and Descriptional Composition

A classical attribute grammar (AG) for a signature Δ is a triple $\mathcal{A} = (I, S, a)$ where for each type $\rho \in \Delta$, $I\rho$ and $S\rho$ are resp. sets of inherited and synthesised attribute values, and for each operator $o \in \Delta^{\rho_1, \cdots, \rho_n \to \rho}$, a_o is a function called the *attribute calculation rule*[1]:

$$a_o : S\rho_1 \times \cdots \times S\rho_n \times I\rho \to S\rho \times I\rho_1 \times \cdots \times I\rho_n. \tag{1}$$

This function captures the input-output relation of a computation unit that processes bidirectional information flow (Figure 1, left). Given a Δ-term M, we connect the assigned computation units according to the shape of M (Figure 1, right). The function corresponding to the entire circuit is the meaning $\mathcal{A}[\![M]\!]$ assigned to M by the AG $\mathcal{A}$. Depending on the configuration of the attribute calculation rules, such function may not exist in general, but if any combination of attribute calculation rules does not yield cyclic information dependency (such AGs are called *non-circular*), the function $\mathcal{A}[\![M]\!]$ uniquely exists for any M. See [14, 15] for the detail.

An *attribute coupling* (AC) from Δ to Σ is a special AG such that the set assigned to $I\rho$ (resp. $S\rho$) is a set $T_\Sigma^{\sigma_1, \cdots, \sigma_n}$ of tuples of Σ-terms for some $\sigma_1, \cdots, \sigma_n \in \Sigma$, and

[1] This form of attribute calculation rule is called *Bochmann normal form*.

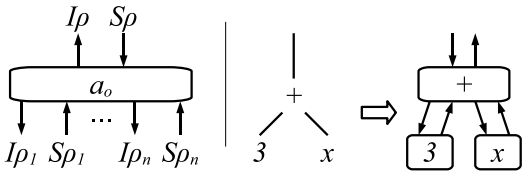

Fig. 1. Attribute Grammars

each attribute calculation rule comprises only of Σ-operators rather than arbitrary functions. Briefly speaking, ACs are AGs constructing Σ-terms. We can extract the essential information from such AGs and redefine ACs from Δ to Σ as the tuple $\mathcal{A} = (I, S, a)$, where for each type $\rho \in \Delta$, $I\rho$ and $S\rho$ are sequences of Σ-types, and for each operator $o \in \Delta^{\rho_1,\cdots,\rho_n \to \rho}$, a_o is a tuple of (open) Σ-terms:

$$a_o \in T_\Sigma^{S\rho, I\rho_1, \cdots, I\rho_n}(S\rho_1, \cdots, S\rho_n, I\rho).$$

We write $\mathbf{AC}(\Delta, \Sigma)$ for the set of ACs from Δ to Σ. We assume that ACs (I, S, a) between the signatures containing the special type # satisfy $I\# = \epsilon$ and $S\# = \#$. With this convention we can view every non-circular AC $\mathcal{A}$ from Δ^+ to Σ^+ as a term transformation function $T\mathcal{A} : T^{\#}_{\Delta^+} \to T^{\#}_{\Sigma^+}$, defined by $T\mathcal{A}(M) = \mathcal{A}[\![M]\!]$.

Let $\mathcal{A} \in \mathbf{AC}(\Delta^+, \Sigma^+)$ and $\mathcal{B} \in \mathbf{AC}(\Sigma^+, \Xi^+)$ be non-circular ACs. We seek for an AC $\mathcal{B} \copyright \mathcal{A}$ such that $T(\mathcal{B} \copyright \mathcal{A}) = T\mathcal{B} \circ T\mathcal{A}$. In general such AC may not exist, but when $\mathcal{A}$ satisfies a condition called *syntactic single-use restriction (SSUR)*, we can build $\mathcal{B} \copyright \mathcal{A}$ by the *descriptional composition*, which we illustrate below.

Suppose that the attribute calculation rules of $\mathcal{A}$ and $\mathcal{B}$ look like the left of Figure 2. There, $\mathcal{A}$ assigns to a Δ^+-operator f a computation unit that constructs Σ^+-terms,

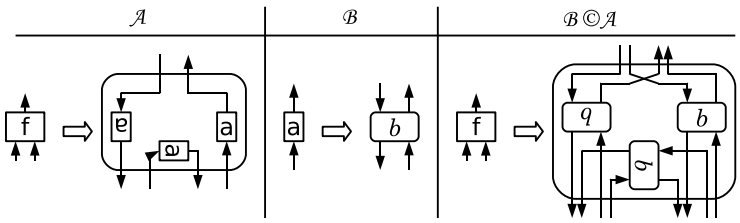

Fig. 2. The descriptional composition

which is drawn as a circuit. Similarly, $\mathcal{B}$ assigns to a Σ^+-operator a a computation unit b, which is just drawn as a round box. We now replace each wire in the right hand side of the attribute calculation rule of $\mathcal{A}$ with bidirectional wire, and replace each Σ^+-term constructor with the computation unit assigned by $\mathcal{B}$. The result of this replacement is drawn on the right of Figure 2, which is a new attribute coupling from Δ^+ to Ξ^+. This is the descriptional composition $\mathcal{B} \copyright \mathcal{A}$.

The hidden point in the above process is that the computation unit (circuit) assigned by $\mathcal{A}$ should not contain any branching wires nor terminals. This is because we do not

know how to make branches and terminals bidirectional (Figure 3). This suggests that

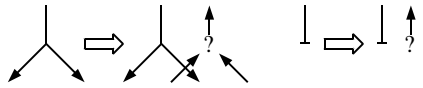

Fig. 3. How to Make Branches and Terminals Bidirectional?

each attribute calculation rule assigned by $\mathcal{A}$ should use each variable exactly once, and an AC satisfying this linearity condition is called *syntactic single use restriction* (*SSUR*) AC. We will see its precise definition in Section 2.2, and reformulate it as an AG in a linear recursive language, which we introduce below.

2.1 The Linear Recursive Language $\mathcal{L}(\Sigma)$

We introduce a simply-typed first-order linear language with recursive declarations called $\mathcal{L}(\Sigma)$. It has only one form of raw-expressions:

$$\lambda x_1, \cdots, x_n \,.\, \mathsf{let}\, y_1 = M_1, \cdots, y_l = M_l \,\mathsf{in}\, z_1, \cdots, z_m,$$

and they are given a type $\sigma_1, \cdots, \sigma_n \to \tau_1, \cdots, \tau_m$ by the type system in Figure 4, where U and D are typing contexts defined by

$$\begin{aligned} \mathrm{U} &= \Gamma_1 \cup \cdots \cup \Gamma_n \cup \{z_1 : \tau_1, \cdots, z_m : \tau_m\} \\ \mathrm{D} &= \{x_1 : \sigma_1, \cdots, x_n : \sigma_n, y_1 : \rho_1, \cdots, y_l : \rho_l\}, \end{aligned}$$

such that x_i, y_j, z_k and variables in $\Gamma_1, \cdots, \Gamma_n$ are different from each other. The leading λ of expressions is a formal binder for $x_1, \cdots, x_n$, rather than the lambda abstraction in the lambda calculus. Expressions are treated modulo α-equivalence.

Expressions of $\mathcal{L}(\Sigma)$ are identified by the rules in Figure 4, where the sequence of variable declarations after let is abbreviated as D. The first axiom allows us to permute D without affecting the meaning of expressions. In the second axiom, $D[w/v]$ denotes the sequence of variable declarations obtained by replacing v in D with w. This axiom allows us to forward v to w when $v = w$ is contained in D.

Here are some examples of $\mathcal{L}(\Sigma_{\mathrm{tree}})$-expressions:

$$\begin{aligned} &\vdash_M \lambda x \,.\, \mathsf{let}\, y = \mathsf{N}(l, x), l = \mathsf{L}\, \mathsf{in}\, y : * \to * \\ &\vdash_M \lambda x, y, z \,.\, \mathsf{let}\, w = \mathsf{N}(x, w), l = \mathsf{L}, v = \mathsf{N}(l, z)\, \mathsf{in}\, y, v : *** \to **. \end{aligned} \tag{2}$$

Note that in (2) the variable w can not be used for output due to the linearity constraint. This means that when the underlying signature has a binary operator then there is a way to discard inputs.

$$\frac{\rho \in \Sigma}{x : \rho \vdash_E x : \rho} \quad \frac{o \in \Sigma^{\rho_1 \cdots \rho_n \to \rho}}{x_1 : \rho_1, \cdots, x_n : \rho_n \vdash_E o(x_1, \cdots, x_n) : \rho}$$

$$\frac{\Gamma_1 \vdash_E M_1 : \rho_1 \quad \cdots \quad \Gamma_l \vdash_E M_l : \rho_l \quad \mathrm{U} = \mathrm{D}}{\vdash_M \lambda \boldsymbol{x} \,.\, \mathsf{let}\, y_1 = M_1, \cdots, y_l = M_l \,\mathsf{in}\, \boldsymbol{z} : \sigma_1, \cdots, \sigma_n \to \tau_1, \cdots, \tau_m}$$

$$(\lambda \boldsymbol{x} \,.\, \mathsf{let}\, D \,\mathsf{in}\, \boldsymbol{z}) = (\lambda \boldsymbol{x} \,.\, \mathsf{let}\, \pi(D) \,\mathsf{in}\, \boldsymbol{z})$$
$$(\lambda \boldsymbol{x} \,.\, \mathsf{let}\, v = w, D \,\mathsf{in}\, \boldsymbol{z}) = (\lambda \boldsymbol{x} \,.\, \mathsf{let}\, D[w/v] \,\mathsf{in}\, \boldsymbol{z}[w/v]) \quad (v \neq w)$$

Fig. 4. Type System and Axiom for $\mathcal{L}(\Sigma)$

2.2 SSUR-ACs as Attribute Grammars in $\mathcal{L}(\Sigma)$

An AC $(I, S, a) \in \mathbf{AC}(\Delta, \Sigma)$ satisfies the *syntactic single use restriction* (*SSUR*) [6] if each attribute calculation rule satisfies the following linearity condition. We let l be the length of the concatenation $S\rho, I\rho_1, \cdots, I\rho_n$, and write τ_i $(1 \leq i \leq l)$ for the i-th component of this sequence. We prepare sequences Γ_i $(1 \leq i \leq l)$ of Σ-types such that $S\rho_1, \cdots, S\rho_n, I\rho$ is a permutation of the concatenation $\Gamma_1, \cdots, \Gamma_l$. We then ask the attribute calculation rule to be in the following set:

$$a_o \in T_\Sigma^{\tau_1}(\Gamma_1) \times \cdots \times T_\Sigma^{\tau_l}(\Gamma_l), \tag{3}$$

and, moreover, in the i-th component of a_o, each variable occurs exactly once.

We observe that there is a one-to-one correspondence between such a tuple and a $\mathcal{L}(\Sigma)$-expression of type $S\rho_1, \cdots, S\rho_n, I\rho \to S\rho, I\rho_1, \cdots, I\rho_n$. We exploit this correspondence to redefine the concept of SSUR-AC.

Definition 1. *An SSUR-AC from Δ to Σ is a triple (I, S, a) where for each type $\rho \in \Delta$, $I\rho$ and $S\rho$ are sequences of Σ-types, and for each $o \in \Delta^{\rho_1, \cdots, \rho_n \to \rho}$, a_o is an $\mathcal{L}(\Sigma)$-expression of type $a_o : S\rho_1, \cdots, S\rho_n, I\rho \to S\rho, I\rho_1, \cdots, I\rho_n$. We write* **SSUR-AC**$(\Delta, \Sigma)$ *for the set of SSUR-ACs from Δ to Σ. When Δ, Σ contains the special type $\#$, we assume that every SSUR-AC (I, S, a) from Δ to Σ satisfies $I\# = \epsilon$ and $S\# = \#$.*

We note that this correspondence is not surjective, as $\mathcal{L}(\Sigma)$ permits recursively defined variables that can not be expressed by SSUR-ACs.

3 Categorical Aspect of Attribute Couplings

3.1 $\mathcal{L}(\Sigma)$ as a Traced Symmetric Monoidal Category

We next view $\mathcal{L}(\Sigma)$ as a category. We regard a sequence $\boldsymbol{\rho} = \rho_1 \cdots \rho_n$ of types in Σ as an object, and an equivalence class of expressions of type $\boldsymbol{\rho} \to \boldsymbol{\sigma}$ as a morphism from $\boldsymbol{\rho}$ to $\boldsymbol{\sigma}$. The composition is defined by

$$(\lambda \boldsymbol{x} \,.\, \mathsf{let}\, D \,\mathsf{in}\, \boldsymbol{y}) \circ (\lambda \boldsymbol{z} \,.\, \mathsf{let}\, D' \,\mathsf{in}\, \boldsymbol{w}) = \lambda \boldsymbol{z} \,.\, \mathsf{let}\, D[\boldsymbol{w}/\boldsymbol{x}], D' \,\mathsf{in}\, \boldsymbol{y}[\boldsymbol{w}/\boldsymbol{x}];$$

here we assume that every bound variable in the above expression is distinct from each other. The category $\mathcal{L}(\Sigma)$ has an evident symmetric strict monoidal structure. The unit object is the empty sequence and the tensor product of two objects is the concatenation of them. The tensor product of two morphisms is defined by merging two expressions:

$$(\lambda \boldsymbol{x} \,.\, \mathsf{let}\ D\ \mathsf{in}\ \boldsymbol{y}) \otimes (\lambda \boldsymbol{z} \,.\, \mathsf{let}\ D'\ \mathsf{in}\ \boldsymbol{w}) = \lambda \boldsymbol{x}, \boldsymbol{z} \,.\, \mathsf{let}\ D, D'\ \mathsf{in}\ \boldsymbol{y}, \boldsymbol{w}.$$

The symmetry morphism is given by $\lambda \boldsymbol{x}, \boldsymbol{y} \,.\, \mathsf{let}\ \epsilon\ \mathsf{in}\ \boldsymbol{y}, \boldsymbol{x}$.

In addition to this, the following *trace operator* constructs recursive declarations in expressions. Let $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{w}$ be sequences of different variables, and $\boldsymbol{\rho}, \boldsymbol{\sigma}, \boldsymbol{\tau}$ be sequences of types such that $|\boldsymbol{x}| = |\boldsymbol{\rho}|, |\boldsymbol{z}| = |\boldsymbol{\sigma}|, |\boldsymbol{y}| = |\boldsymbol{w}| = |\boldsymbol{\tau}|$. We define the *trace operator* $\mathrm{tr}^{\boldsymbol{\tau}}_{\boldsymbol{\rho},\boldsymbol{\sigma}}$ as follows:

$$\mathrm{tr}^{\boldsymbol{\tau}}_{\boldsymbol{\rho},\boldsymbol{\sigma}}(\lambda \boldsymbol{x}, \boldsymbol{y} \,.\, \mathsf{let}\ D\ \mathsf{in}\ \boldsymbol{z}, \boldsymbol{w}) = \lambda \boldsymbol{x} \,.\, \mathsf{let}\ \boldsymbol{y} = \boldsymbol{w}, D\ \mathsf{in}\ \boldsymbol{z}.$$

A pair of a symmetric monoidal category and a trace operator is called *traced symmetric monoidal category* (see Appendix B). In this article we only consider the traced symmetric *strict* monoidal categories, and call them TSMC. [2] The above discussion is summarised as follows:

Proposition 1. *The above data make $\mathcal{L}(\Sigma)$ a TSMC.*

We call a symmetric strong monoidal functor between TSMCs *traced* if it preserves the trace operator in an evident way (see Appendix B). We write **TSMC** for the 2-category of TSMCs, traced symmetric strong monoidal functors and monoidal natural isomorphisms. Its 2-subcategory consisting of traced symmetric strict monoidal functors will be denoted by $\mathbf{TSMC}_s$.

From a traced strong monoidal functor $(F, \phi^F_\epsilon, \phi^F_{\boldsymbol{\rho},\boldsymbol{\sigma}}) \in \mathbf{TSMC}(\mathcal{L}(\Sigma), \mathbb{C})$, we can construct a traced strict monoidal functor $\mathbf{Str}(F) \in \mathbf{TSMC}_s(\mathcal{L}(\Sigma), \mathbb{C})$ which is naturally isomorphic to F. The strict functor is constructed as follows:

$$\mathbf{Str}(F)(\rho_1 \cdots \rho_n) = F\rho_1 \otimes \cdots \otimes F\rho_n \qquad \mathbf{Str}(F)(f : \boldsymbol{\rho} \to \boldsymbol{\sigma}) = (\phi^F_{\boldsymbol{\sigma}})^{-1} \circ Ff \circ \phi^F_{\boldsymbol{\rho}}.$$

The assignment $F \mapsto \mathbf{Str}(F)$, which we will call *strictification operator*, extends to an equivalence of categories $\mathbf{Str} : \mathbf{TSMC}(\mathcal{L}(\Sigma), \mathbb{C}) \to \mathbf{TSMC}_s(\mathcal{L}(\Sigma), \mathbb{C})$. It satisfies the following property: for any $F \in \mathbf{TSMC}_s(\mathcal{L}(\Sigma), \mathbb{D}), G \in \mathbf{TSMC}(\mathbb{D}, \mathbb{E}), H \in \mathbf{TSMC}(\mathbb{E}, \mathbb{F})$, we have $\mathbf{Str}(F) = F$ and $\mathbf{Str}(H \circ G \circ F) = \mathbf{Str}(H \circ \mathbf{Str}(G \circ F))$.

3.2 Monoidal Attribute Grammar

We generalise the underlying semantic domain of AGs to arbitrary $\mathbb{C} \in \mathbf{TSMC}$. This generalisation is done by replacing sets with objects and attribute calculation rules with $\mathbb{C}$-morphisms.

Definition 2 ([12]). *Let $\mathbb{C} \in \mathbf{TSMC}$. A* monoidal attribute grammar *(MAG) for Δ in $\mathbb{C}$ is a triple (I, S, a) where for each type $\rho \in \Sigma$, $I\rho$ and $S\rho$ are $\mathbb{C}$-objects (of domains of inherited and synthesised attributes), and for each operator $o \in \Delta^{\rho_1 \cdots \rho_n \to \rho}$, a_o is a $\mathbb{C}$-morphism of type $a_o : S\rho_1 \otimes \cdots \otimes S\rho_n \otimes I\rho \to S\rho \otimes I\rho_1 \otimes \cdots \otimes I\rho_n$. We write $\mathbf{MAG}(\Delta, \mathbb{C})$ for the collection of MAGs for Δ in $\mathbb{C}$.*

[2] Every traced symmetric monoidal category is equivalent to a traced symmetric strict monoidal category (coherence theorem).

Some instances of MAGs are studied in [12]; in the category ω**CPPO** of pointed CPOs and ω-continuous functions, monoidal attribute grammars are equivalent to Chirica and Martin's K-systems [4]. The category **Rel** of sets and relations has traced biproducts [11], and MAGs in this traced symmetric monoidal category are *local dependency graphs*, which are the standard tool to represent dependencies between the attributes in attribute calculation rules. MAGs over the compact closed structure on **Rel** are *relational attribute grammars* [5]. In addition to this, by comparing Definition 1 and 2, we conclude that SSUR-ACs are exactly MAGs in $\mathcal{L}(\Sigma)$.

Proposition 2. **SSUR-AC**(Δ, Σ) = **MAG**$(\Delta, \mathcal{L}(\Sigma))$.

3.3 MAGs as Algebras in Int($\mathbb{C}$)

Below we give two concepts that are equivalent to MAG: one is algebras in the categories obtained by Joyal, Street and Verity's **Int** construction [11], and the other is traced symmetric strict monoidal functors of type $\mathcal{L}(\Sigma) \to$ **Int**$(\mathbb{C})$.

Let $\mathbb{C} \in$ **TSMC**. The category **Int**$(\mathbb{C})$ is defined by the following data: an object is a pair (A^+, A^-) of $\mathbb{C}$-objects, and homsets are defined by

$$\mathbf{Int}(\mathbb{C})((A^+, A^-), (B^+, B^-)) = \mathbb{C}(A^+ \otimes B^-, B^+ \otimes A^-).$$

In category **Int**$(\mathbb{C})$ we can naturally model computation over bidirectional information flow. An object (A^+, A^-) denotes the type of upward and downward information; in the context of attribute grammar, they correspond to the type of synthesised and inherited attributes, respectively. A morphism $f : (A^+, A^-) \to (B^+, B^-)$ then represents a computation that processes bidirectional information flow.

We give a symmetric strict monoidal structure to **Int**$(\mathbb{C})$ by

$$\mathbf{I}^{\mathbf{Int}(\mathbb{C})} = (\mathbf{I}^{\mathbb{C}}, \mathbf{I}^{\mathbb{C}}), \qquad (A^+, A^-) \otimes^{\mathbf{Int}(\mathbb{C})} (B^+, B^-) = (A^+ \otimes^{\mathbb{C}} B^+, A^- \otimes^{\mathbb{C}} B^-).$$

The category **Int**$(\mathbb{C})$ has a *compact closed structure* [11], which yields the *canonical trace operator* with respect to the above symmetric monoidal structure. The mapping $\mathbb{C} \mapsto$ **Int**$(\mathbb{C})$ extends to a 2-endofunctor over **TSMC**, and moreover, to a pseudo-monad $(\mathbf{Int}, (N, n), (M, m), \tau, \lambda, \rho)$ over **TSMC**; see Appendix B for the detail.

We extend the concept of Σ-algebra from the set-theoretic one to the categorical one. A Σ-algebra in a monoidal category $\mathbb{C}$ is a pair (A, a) where A is a family of $\mathbb{C}$-objects indexed by Σ-types and a is a family of $\mathbb{C}$-morphisms indexed by Σ-operators, such that the type of a_o is $A\rho_1 \otimes \cdots \otimes A\rho_n \to A\rho$ for each operator $o \in \Sigma^{\rho_1 \cdots \rho_n \to \rho}$. We write $\mathbf{Alg}_\Sigma(\mathbb{C})$ for the collection of Σ-algebras in $\mathbb{C}$. The concept of Σ-algebras has another presentation: there is a natural bijection between Σ-algebras in $\mathbb{C}$ and traced symmetric strict monoidal functors from $\mathcal{L}(\Sigma)$ to $\mathbb{C}$:

$$\mathbf{Alg}_\Sigma(\mathbb{C}) \simeq \mathbf{TSMC}_s(\mathcal{L}(\Sigma), \mathbb{C}) \tag{4}$$

Let (I, S, a) be a MAG for Δ in $\mathbb{C} \in$ **TSMC**. We define $A\rho$ to be the pair $(S\rho, I\rho)$ of $\mathbb{C}$-objects (note that it is an object in **Int**$(\mathbb{C})$). Then for each operator $o \in \Sigma^{\rho_1 \cdots \rho_n \to \rho}$, the $\mathbb{C}$-morphism a_o can be seen as an **Int**$(\mathbb{C})$-morphism:

$$a_o \in \mathbb{C}(S\rho_1 \otimes \cdots \otimes S\rho_n \otimes I\rho, S\rho \otimes I\rho_1 \otimes \cdots \otimes I\rho_n) = \mathbf{Int}(\mathbb{C})(A\rho_1 \otimes \cdots \otimes A\rho_n, A\rho).$$

This means that every MAG determines a Δ-algebra (A, a) in $\mathbf{Int}(\mathbb{C})$, and the other way around. We summarise these concepts by the following bijective correspondences:

$$\mathbf{MAG}(\Delta, \mathbb{C}) \simeq \mathbf{Alg}_{\Delta}(\mathbf{Int}(\mathbb{C})) \simeq \mathbf{TSMC}_s(\mathcal{L}(\Delta), \mathbf{Int}(\mathbb{C})) \tag{5}$$

$$\mathbf{SSUR\text{-}AC}(\Delta, \Sigma) \simeq \mathbf{TSMC}_s(\mathcal{L}(\Delta), \mathbf{Int}(\mathcal{L}(\Sigma))). \tag{6}$$

These three equivalent forms have different advantages. The first form is the actual data we give when defining AGs. The second form is used to explain the initial algebra semantics of AGs [12]. The third form is suitable for discussing the descriptional composition. We mainly adopt the functorial representation of AGs and ACs below.

4 Descriptional Composition

We begin with a categorical formulation of the descriptional composition of SSUR-ACs. Let $\mathcal{A} \in \mathbf{SSUR\text{-}AC}(\Delta, \Sigma)$ and $\mathcal{B} \in \mathbf{MAG}(\Sigma, \mathbb{C})$, regarded as functors. We define their (categorical) descriptional composition $\mathcal{B} \copyright \mathcal{A}$ by

$$\mathcal{B} \copyright \mathcal{A} = \mathbf{Str}(\mathcal{B}^{\#} \circ \mathcal{A}),$$

where $\mathcal{B}^{\#} = M_{\mathbb{C}} \circ \mathbf{Int}(\mathcal{B})$ is the Kleisli lifting of $\mathcal{B}$ by the pseudo-monad **Int**. We insert the strictification operator **Str** (Section 3.1) as $\mathcal{B}^{\#}$ is not strict monoidal.[3] The SSUR-AC $\mathcal{A}$ constructs bidirectional networks of Σ-operators, while $\mathcal{B}$ can only accept single-directional networks of them. The Kleisli lifting extends the domain of $\mathcal{B}$ to the bidirectional network of Σ-operators (see Figure of 5) so that this mismatch is resolved.

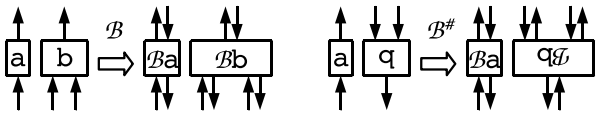

Fig. 5. Kleisli Lifting of $\mathcal{B}$

Theorem 1. *1. SSUR-ACs are closed under the descriptional composition.*

2. The descriptional composition is associative up to a natural isomorphisms; for any $\mathcal{A} \in \mathbf{SSUR\text{-}AC}(\Delta, \Sigma), \mathcal{B} \in \mathbf{SSUR\text{-}AC}(\Sigma, \Xi)$ and $C \in \mathbf{MAG}(\Xi, \mathbb{C})$, there is a natural isomorphism between $(C \copyright \mathcal{B}) \copyright \mathcal{A}$ and $C \copyright (\mathcal{B} \copyright \mathcal{A})$.

We do not prove this theorem as it is subsumed by Theorem 2. We note that the associativity of the descriptional composition holds only up to a natural isomorphism. This isomorphism has no computational meaning; it just permutes the order of arguments. This permutation is invisible in the syntactic study of the descriptional composition because arguments are passed by records rather than tuples.

[3] This is because the multiplication $M_{\mathbb{C}} : \mathbf{Int}^2(\mathbb{C}) \to \mathbf{Int}(\mathbb{C})$ is not strict monoidal.

We extend this formulation of the descriptional composition to a more general setting where TSMCs are equipped with some extra structures, such as nondeterminism, undefined values, stacks, etc. To capture such extensions, we introduce the concept of *extension of TSMC*.

Definition 3. *We call the following situation an* extension of TSMC.

1. *There is a 2-category* $\mathbf{TSMC}'$ *and its 2-subcategory* $\mathbf{TSMC}'_s$.
2. *There is a 2-functor* $U : \mathbf{TSMC}' \to \mathbf{TSMC}$ *that can be restricted to a 2-functor* $U_s : \mathbf{TSMC}'_s \to \mathbf{TSMC}_s$. *Furthermore,* U_s*, as an ordinary functor, has a left adjoint* F. *We write* $\overline{(-)} : |\mathbf{TSMC}(F\mathbb{C}, \mathbb{D})| \to |\mathbf{TSMC}(\mathbb{C}, U_s\mathbb{D})|$ *for the bijection between homsets, and* $\underline{(-)}$ *for its inverse.*
3. *There is a pseudo-monad* $(\mathbf{Int}', (N', n'), (M', m'), \tau', \lambda', \rho')$ *over* $\mathbf{TSMC}'$ *such that* $\mathbf{Int}'$ *is a 2-functor, and is a strict lifting of the pseudo-monad* $\mathbf{Int}$ *along* U*, that is,* $U \circ \mathbf{Int}' = \mathbf{Int} \circ U$, $U(N'_F) = N_{UF}$, $U\tau'_{\mathbb{C}} = \tau_{U\mathbb{C}}$, *etc. We note that the Kleisli lifting* $(-)^\flat$ *of* $\mathbf{Int}'$ *also commutes with* U*, that is,* $U((\mathcal{A})^\flat) = (U\mathcal{A})^\#$.
4. *There is an equivalence* $\mathbf{Str}' : \mathbf{TSMC}'(F\mathcal{L}(\Sigma), \mathbb{C}) \to \mathbf{TSMC}'_s(F\mathcal{L}(\Sigma), \mathbb{C})$ *such that* $\overline{\mathbf{Str}'(F)} = \mathbf{Str}(\overline{F})$*, and for any functor* $F \in \mathbf{TSMC}'_s$ *and* $G, H \in \mathbf{TSMC}'$ *of appropriate type,* $\mathbf{Str}'(F) = F$ *and* $\mathbf{Str}'(H \circ G \circ F) = \mathbf{Str}'(H \circ \mathbf{Str}'(G \circ F))$.

We express such a situation by a tuple $\mathcal{E} = (F, U, \mathbf{Int}')$.

Definition 4. *Let* $\mathcal{E} = (F, U, \mathbf{Int}')$ *be an extension of TSMC. We define the collection of* $\mathcal{E}$*-MAG for* Δ *in* $\mathbb{C} \in \mathbf{TSMC}'$ *and* $\mathcal{E}$*-AC from* Δ *to* Σ *by*

$$\mathcal{E}\text{-}\mathbf{MAG}(\Delta, \mathbb{C}) = \mathbf{TSMC}_s(\mathcal{L}(\Delta), \mathbf{Int}(U\mathbb{C})) \simeq \mathbf{MAG}(\Delta, U\mathbb{C})$$
$$\mathcal{E}\text{-}\mathbf{AC}(\Delta, \Sigma) = \mathbf{TSMC}_s(\mathcal{L}(\Delta), \mathbf{Int}(UF\mathcal{L}(\Sigma))) \simeq \mathbf{MAG}(\Delta, UF\mathcal{L}(\Sigma)).$$

Let $\mathcal{A} \in \mathcal{E}\text{-}\mathbf{AC}(\Delta, \Sigma)$ and $\mathcal{B} \in \mathcal{E}\text{-}\mathbf{MAG}(\Sigma, \mathbb{C})$. We define their descriptional composition $\mathcal{B} \copyright \mathcal{A}$ by

$$\mathcal{B} \copyright \mathcal{A} = \mathbf{Str}(U((\underline{\mathcal{B}})^\flat) \circ \mathcal{A}).$$

We write $H_{\mathbb{C}} : \mathbb{C} \to FU_s\mathbb{C}$ for the unit of the adjunction $F \dashv U_s$. We assume that any $\mathcal{E}$-AC $\mathcal{A}$ between rooted signatures satisfies $\mathcal{A}(\#) = (\mathbf{I}, H(\#))$. We define the translation $T\mathcal{A} : UF\mathcal{L}(\Delta^+)(\mathbf{I}, H\#) \to UF\mathcal{L}(\Sigma^+)(\mathbf{I}, H\#)$ induced by $\mathcal{A} \in \mathcal{E}\text{-}\mathbf{AC}(\Delta^+, \Sigma^+)$ as follows:

$$T\mathcal{A}(f) = \text{unique } g \text{ such that } U\underline{\mathcal{A}}(f) = N_{UF\mathcal{L}(\Sigma)}(g).$$

This is well-defined as N is full and faithful [11]. When we do not consider the extension ($F = U = \mathrm{Id}$), the translation $T\mathcal{A}$ is just a mapping of $f \in \mathcal{L}(\Delta^+)(\epsilon, \#)$ to a morphism in $\mathcal{L}(\Sigma^+)(\epsilon, \#)$. Under the identification of terms and morphisms in a free TSMC, $T\mathcal{A}$ represents the term translation induced by the attribute coupling $\mathcal{A}$.

Theorem 2. *Let* $\mathcal{E}$ *be an extension of TSMC.*

1. $\mathcal{E}$*-ACs are closed under descriptional composition.*
2. *For any* $\mathcal{A} \in \mathcal{E}\text{-}\mathbf{AC}(\Delta, \Sigma), \mathcal{B} \in \mathcal{E}\text{-}\mathbf{AC}(\Sigma, \Xi)$ *and* $C \in \mathcal{E}\text{-}\mathbf{MAG}(\Xi, \mathbb{C})$*, there is a natural isomorphism between* $(C \copyright \mathcal{B}) \copyright \mathcal{A}$ *and* $C \copyright (\mathcal{B} \copyright \mathcal{A})$.

3. *For any* $\mathcal{A} \in \mathcal{E}\text{-}\mathbf{AC}(\Delta^+, \Sigma^+), \mathcal{B} \in \mathcal{E}\text{-}\mathbf{AC}(\Sigma^+, \Xi^+)$ *and* $f \in \mathcal{L}(\Delta^+)(\epsilon, \#)$*, we have* $T(\mathcal{B} \copyright \mathcal{A}) \circ H_{\mathcal{L}(\Delta^+)}(f) = T\mathcal{B} \circ T\mathcal{A} \circ H_{\mathcal{L}(\Delta^+)}(f)$.

The proof is in Appendix A.

Corollary 1. *For any composable SSUR-ACs* $\mathcal{A}, \mathcal{B}$*, we have* $T(\mathcal{B} \copyright \mathcal{A}) = T\mathcal{B} \circ T\mathcal{A}$.

In the subsequent sections, we show that some useful extensions of (SSUR)-ACs can be captured as attribute couplings in extensions of TSMC. From the above general theorem, the associativity of the descriptional composition and the closure property of extended ACs under the descriptional composition.

4.1 Descriptional Composition for Nondeterministic ACs

We look at an example of an extension of TSMC arising from a symmetric monoidal monad $(T, \eta, \mu, \phi_{\mathbf{I}}, \phi_{A,B})$ over **Set**. [4] Given such a monad, for a category $\mathbb{C}$, we define a new category $T_*(\mathbb{C})$ by the following data: $|T_*(\mathbb{C})| = |\mathbb{C}|$ and $T_*(\mathbb{C})(A, B) = T(\mathbb{C}(A, B))$. The identity and composition of $T_*(\mathbb{C})$ defined as follows:

$$1 \xrightarrow{\phi_{\mathbf{I}}} T1 \xrightarrow{T(\mathrm{id}^{\mathbb{C}}_A)} T_*(\mathbb{C})(A, A)$$

$$T_*(\mathbb{C})(B, C) \times T_*(\mathbb{C})(A, B) \xrightarrow{\phi_{\mathbb{C}(B,C),\mathbb{C}(A,B)}} T(\mathbb{C}(B, C) \times \mathbb{C}(A, B)) \xrightarrow{T(comp^{\mathbb{C}})} T_*(\mathbb{C})(A, C).$$

This construction is well-known as *change-of-base* in enriched category theory. The mapping $\mathbb{C} \mapsto T_*\mathbb{C}$ extends to a 2-monad (T_*, μ_*, η_*) over **TSMC** and $\mathbf{TSMC}_s$. We write $\mathbf{TSMC}^{T_*}$ for the 2-category of strict T_*-algebras (which are exactly T-algebra enriched TSMCs), strict T_*-algebra morphisms and T_*-algebra transformations; see [2] for the detail. We also write $\mathbf{TSMC}_s^{T_*}$ for its 2-subcategory such that T_*-algebra morphisms belong to $\mathbf{TSMC}_s$. The Eilenberg-Moore 2-adjunction $F \dashv U : \mathbf{TSMC}^{T_*} \to \mathbf{TSMC}$ can be restricted to $F_s \dashv U_s : \mathbf{TSMC}_s^{T_*} \to \mathbf{TSMC}_s$. One can easily extend the strictification operator **Str** to the one satisfying the condition 4 of Definition 3.

To lift the pseudo-monad **Int** to the 2-category $\mathbf{TSMC}^{T_*}$, we (necessarily) give a pseudo-distributive law [21] of T_* over **Int**. In fact, it consists only of identities, as the components of **Int** and T_* commutes with each other, such as $\mathbf{Int}(T_*(\mathbb{C})) = T_*(\mathbf{Int}(\mathbb{C}))$, $T_*(N_{\mathbb{C}}) = N_{T_*\mathbb{C}}$, $\mathbf{Int}((\mu_*)_{\mathbb{C}}) = (\mu_*)_{\mathbf{Int}(\mathbb{C})}$, etc. Thus the above data determine an extension of TSMC.

Example 1. We write $\mathcal{P} : \mathbf{Set} \to \mathbf{Set}$ for the covariant powerset monad. Let $\mathbb{C} \in \mathbf{TSMC}$. A monoidal attribute grammar (I, S, a) for Σ in $\mathcal{P}_*(\mathbb{C})$ assigns $\mathbb{C}$-objects $I\rho, S\rho$ to each type $\rho \in \Sigma$, and a $\mathcal{P}_*(\mathbb{C})$-morphism

$$a_o \in \mathcal{P}_*(\mathbb{C})(S\rho_1 \otimes \cdots \otimes S\rho_n \otimes I\rho, S\rho \otimes I\rho_1 \otimes \cdots \otimes I\rho_n)$$

to each operator $o \in \Sigma^{\rho_1 \cdots \rho_n \to \rho}$; this means that a $\mathcal{P}_*$-MAG assigns an attribute calculation rule *nondeterministically* to each operator in Σ. For instance, we consider the following $\mathcal{P}_*$-AC from Σ_{nat} to Σ_{list}:

$$I(*) = \epsilon, \qquad S(*) = *,$$

[4] This is equivalent to a commutative monad [16].

$$a_{\mathsf{Z}} = \{[]\}, \qquad a_{\mathsf{S}} = \{(\lambda x\,.\,\mathsf{let}\ y = 0 :: x\ \mathsf{in}\ y), (\lambda x\,.\,\mathsf{let}\ y = 1 :: x\ \mathsf{in}\ y)\}.$$

This AC maps each natural number $\mathsf{S}^{(n)}(\mathsf{Z})$ to the set of all binary digit with length n, represented as morphisms of type $\epsilon \to *$ in $\mathcal{P}_*(\mathcal{L}(\Sigma_{\mathrm{list}}))$.

4.2 Descriptional Composition for quasi-SSUR ACs

In [20], Nishimura and Nakano introduced a relaxation of SSUR called *quasi-SSUR*. It relaxes the linear use (exactly once) of variables to affine use (at most once), and introduces a constant denoting the undefined value to the language for attribute calculation rules. We formulate their quasi-SSUR ACs and the descriptional composition of them in our categorical framework.

First, we introduce the language $\mathcal{A}(\Sigma)$ that modifies the typing rules of $\mathcal{L}(\Sigma)$ so that variables can be discarded, and that has an extra constant $\bot_\rho$ for each type $\rho \in \Sigma$. These features directly correspond to Nishimura and Nakano's modification of SSUR. The detail of $\mathcal{A}(\Sigma)$ is the following:

1. We replace the typing rule of $\mathcal{L}(\Sigma)$ in Figure 4 as follows:

$$\frac{\Gamma_1 \vdash_E M_1 : \rho_1 \quad \cdots \quad \Gamma_l \vdash_E M_l : \rho_l \quad \mathrm{U} \subseteq \mathrm{D}}{\vdash_M \lambda \boldsymbol{x}\,.\,\mathsf{let}\ y_1 = M_1, \cdots, y_l = M_l\ \mathsf{in}\ \boldsymbol{z} : \sigma_1, \cdots, \sigma_n \to \tau_1, \cdots, \tau_m}$$

   What is new to $\mathcal{L}(\Sigma)$ is that some defined variables may be unused ($\mathrm{U} \subseteq \mathrm{D}$). For instance, the following is now a valid derivation in $\mathcal{A}(\Sigma)$:

$$\frac{}{\vdash_M \lambda x_1 \cdots x_n\,.\,\mathsf{let}\ \epsilon\ \mathsf{in}\ \epsilon : \rho_1 \cdots \rho_n \to \epsilon}$$

   We write this expression $\top_{\rho_1 \cdots \rho_n}$.
2. We add to $\mathcal{L}(\Sigma)$ a constant $\bot_\rho : \rho$ for each type $\rho \in \Sigma$, corresponding to undef in [20]. We then extend this to any sequence of Σ-types by

$$\bot_{\rho_1 \cdots \rho_n} = (\lambda \epsilon\,.\,\mathsf{let}\ x_1 = \bot_{\rho_1}, \cdots, x_n = \bot_{\rho_n}\ \mathsf{in}\ x_1, \cdots, x_n).$$

Definition 5. *A quasi-SSUR AC from Δ to Σ is a triple (I, S, a) where for each type $\rho \in \Delta$, $I\rho$ and $S\rho$ are sequences of Σ-types, and for each $o \in \Delta^{\rho_1, \cdots \rho_n \to \rho}$, a_o is an $\mathcal{A}(\Sigma)$-expression of type $a_o : S\rho_1, \cdots, S\rho_n, I\rho \to S\rho, I\rho_1, \cdots, I\rho_n$.*

We next introduce the concept of *bipointed TSMC*. It is a triple $(\mathbb{C}, \top, \bot)$ where $\mathbb{C} \in \mathbf{TSMC}$ and $\top, \bot$ are $\mathbb{C}$-object indexed families of morphisms $\top_A : A \to \mathbf{I}$ and $\bot_A : \mathbf{I} \to A$ such that

$$\top_{\mathbf{I}} = \mathrm{id}_{\mathbf{I}}, \qquad \top_{A \otimes B} = \top_A \otimes \top_B, \qquad \bot_{\mathbf{I}} = \mathrm{id}_{\mathbf{I}}, \qquad \bot_{A \otimes B} = \top_A \otimes \top_B. \tag{7}$$

Discarding variables is modelled by the morphism $\top$. We say that a traced symmetric strong monoidal functor $(F, \phi_{\mathbf{I}}, \phi_{A,B})$ *preserves bipoints* if it satisfies $F(\bot_A) \circ \phi_{\mathbf{I}} = \bot_{FA}$ and $\phi_{\mathbf{I}} \circ \top_{FA} = F(\top_A)$. We write $\mathbf{TSMC}^\bullet$ for the 2-category of bipointed TSMCs, traced symmetric strong monoidal functors preserving bipoints and monoidal natural isomorphisms. We write $\mathbf{TSMC}^\bullet_s$ for its 2-subcategory consisting of traced symmetric

strict monoidal functors. There is an evident forgetful functor $U^\bullet : \mathbf{TSMC}^\bullet \to \mathbf{TSMC}$ that can be restricted to $U^\bullet_s : \mathbf{TSMC}^\bullet_s \to \mathbf{TSMC}_s$, and it has an ordinary left adjoint $F^\bullet : \mathbf{TSMC}_s \to \mathbf{TSMC}^\bullet_s$ that freely adds morphisms $\bot_A : \mathbf{I} \to A$ and $\top_A : A \to \mathbf{I}$, subject to the equations in (7).

The **Int** construction can be lifted over $\mathbf{TSMC}^\bullet$. For a bipointed TSMC $(\mathbb{C}, \bot, \top)$, we define $\mathbf{Int}(\mathbb{C})$-morphisms $\bot_A^{\mathbf{Int}(\mathbb{C})}$ and $\top_A^{\mathbf{Int}(\mathbb{C})}$ by

$$\bot_A^{\mathbf{Int}(\mathbb{C})} = \bot_{A^+} \otimes \top_{A^-}, \qquad \top_A^{\mathbf{Int}(\mathbb{C})} = \top_{A^+} \otimes \bot_{A^-}.$$

Then we define $\mathbf{Int}^\bullet(\mathbb{C}, \bot, \top)$ to be the tuple $(\mathbf{Int}(\mathbb{C}), \bot^{\mathbf{Int}(\mathbb{C})}, \top^{\mathbf{Int}(\mathbb{C})})$. One can easily check that this is a bipointed TSMC. The mapping $(\mathbb{C}, \bot, \top) \mapsto \mathbf{Int}^\bullet(\mathbb{C}, \bot, \top)$ extends to a 2-functor over $\mathbf{TSMC}^\bullet$, and the pseudo-monad structure of **Int** also makes $\mathbf{Int}^\bullet$ a pseudo-monad. Furthermore, it is a strict lifting of **Int** along U.

Proposition 3. *The triple $\mathcal{E}^\bullet = (F^\bullet, U^\bullet, \mathbf{Int}^\bullet)$ forms an extension of TSMCs.*

Proposition 4. *The tuple $(\mathcal{A}(\Sigma), \bot, \top)$ is a bipointed TSMC, and it is equivalent to $F(\mathcal{L}(\Sigma))$. Therefore, quasi-SSUR ACs are $\mathcal{E}^\bullet$-ACs.*

4.3 Descriptional Composition for Stack AGs

In [18], Nakano introduced an extension of AC called *stack AC*, where we can use stacks in attribute calculation rules. He then showed that the stack ACs satisfying certain linearity condition are closed under the descriptional composition. Inspired by his extension, below we express his theory of stack ACs in our categorical framework by setting-up an appropriate extension of TSMC. Our approach differs from Nakano's work as follows: 1) the concept of stack ACs given below allow stacks to be stack elements, and 2) we represent the empty stack by undefined stack $\bot$, and combine stack deconstructors (head, tail) into single operator dec.

We introduce a language $\mathcal{S}(\Sigma)$ and capture the stack ACs satisfying a linearity condition as AGs in $\mathcal{S}(\Sigma)$. In this language we can use a stack of type ρ, whose type is denoted by ρ^∞. Stacks are then manipulated by two operators: constructor $\mathsf{cons} : \rho, \rho^\infty \to \rho^\infty$ and deconstructor $\mathsf{dec} : \rho^\infty \to \rho, \rho^\infty$. The operator cons pushes a given value on a given stack, while dec separates the top value of a given stack from the rest of it.

We move on to the formal definition of $\mathcal{S}(\Sigma)$. First, we define $\mathcal{S}(\Sigma)_0$ to be the set of pairs of the form (n, ρ) where n is a natural number and ρ is a Σ-type. We identify a type $\rho \in \Sigma$ and the pair $(0, \rho) \in \mathcal{S}(\Sigma)_0$. We denote a pair (n, ρ) by $\rho^{\infty \cdots \infty}$, where n is the length of ∞'s on the shoulder. Next, we define $|\mathcal{S}(\Sigma)|$ to be the set of finite sequences of $\mathcal{S}(\Sigma)_0$. Below we use metavariables B and C to denote elements in $\mathcal{S}(\Sigma)_0$ and $|\mathcal{S}(\Sigma)|$, respectively. The set of raw terms of $\mathcal{S}(\Sigma)$ is defined by

$$M ::= \lambda \boldsymbol{x}.\, \mathrm{let}\ \boldsymbol{D}\ \mathrm{in}\ \boldsymbol{x}$$
$$D ::= x = x \mid x = o(\boldsymbol{x}) \mid x = \bot_B \mid x = \mathsf{cons}_B(x, x) \mid x, x = \mathsf{dec}_B(x).$$

The typing rules of $\mathcal{S}(\Sigma)$ extends the one for $\mathcal{A}(\Sigma)$ with cons and dec:

$$\frac{}{x : B \vdash_E x : B} \qquad \frac{}{\vdash_E \bot_B : B} \qquad \frac{o \in \Sigma^{\rho_1 \cdots \rho_n \to \rho}}{x_1 : \rho_1, \cdots, x_n : \rho_n \vdash_E o(x_1, \cdots, x_n) : \rho}$$

$$\frac{}{x : B, y : B^\infty \vdash_E \mathsf{cons}_B(x,y) : B^\infty} \qquad \frac{}{x : B^\infty \vdash_E \mathsf{dec}_B(x) : B, B^\infty}$$

$$\frac{\Gamma_1 \vdash_E M_1 : C_1 \quad \cdots \quad \Gamma_l \vdash_E M_l : C_l \quad \mathrm{U} \subseteq \mathrm{D}}{\vdash_M \lambda \boldsymbol{x} \,.\, \mathsf{let}\ \boldsymbol{y}_1 = M_1, \cdots, \boldsymbol{y}_l = M_l\ \mathsf{in}\ \boldsymbol{z} : C' \to C''}$$

where $\mathrm{U} = \Gamma_1 \cup \cdots \cup \Gamma_l \cup \{\boldsymbol{z} : C''\}$ and $\mathrm{D} = \{\boldsymbol{x} : C', \boldsymbol{y}_1 : C_1, \cdots, \boldsymbol{y}_l : C_l\}$, and each variable in $\Gamma_1, \cdots, \Gamma_n, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ is different from the other. The set of axioms for $\mathcal{S}(\Sigma)$-expressions extends the one for $\mathcal{A}(\Sigma)$ with the following rules (leading λ is omitted):

$$\begin{aligned}
(\mathsf{let}\ x, y = \mathsf{dec}_B(z), z = \mathsf{cons}_B(x', y')\ \mathsf{in}\ \boldsymbol{v}) &= (\mathsf{let}\ x = x', y = y'\ \mathsf{in}\ \boldsymbol{v}) \\
(\mathsf{let}\ x, y = \mathsf{dec}_B(z), z = \bot_{B^\infty}\ \mathsf{in}\ \boldsymbol{v}) &= (\mathsf{let}\ x = \bot_B, y = \bot_{B^\infty}\ \mathsf{in}\ \boldsymbol{v}) \\
(\mathsf{let}\ z = \mathsf{cons}_B(x,y)\ \mathsf{in}\ \boldsymbol{v}) &= (\mathsf{let}\ D\ \mathsf{in}\ \boldsymbol{v} \quad (z \notin FV(D) \cup \boldsymbol{v} \cup \{y\})).
\end{aligned}$$

Definition 6. *A quasi-SSUR stack AC from Δ to Σ is a triple (I, S, a) where for each type $\rho \in \Delta$, $I\rho, S\rho \in |\mathcal{S}(\Sigma)|$, and for each operator $o \in \Delta^{\rho_1, \cdots, \rho_n \to \rho}$, a_o is an $\mathcal{S}(\Sigma)$-expression of type $a_o : S\rho_1, \cdots, S\rho_n, I\rho \to S\rho, I\rho_1, \cdots, I\rho_n$.*

Example 2. This example is from [18]. We consider a stack AC that converts reverse-polish notations to ordinary expressions. Let Σ_p and Σ_e be signatures defined as follows:

$$\begin{aligned}
\Sigma_p &= (\{*\}, \{\mathsf{p}_n^{*\to *}, \mathsf{a}^{*\to *}, \mathsf{m}^{*\to *}, \mathsf{r}^{\to *}\}) \quad (n \in \mathbf{N}), \\
\Sigma_e &= (\{*\}, \{\mathsf{num}_n^{\to *}, \mathsf{add}^{*,*\to *}, \mathsf{mul}^{*,*\to *}\}) \quad (n \in \mathbf{N}).
\end{aligned}$$

The signature Σ_p is for the reverse-polish notation of expressions. For instance, a Σ_p-expression $\mathsf{p}_3(\mathsf{p}_2(\mathsf{p}_5(\mathsf{a}(\mathsf{m}(\mathsf{r})))))$ denotes $(5+2)*3$. The stack AC (I, S, a) that constructs Σ_e-terms from reverse-polish expressions is the following (type annotations are omitted):

$$\begin{aligned}
I* &= (*)^\infty, O* = * \\
a_{\mathsf{p}_n} &= \lambda s_1, i \,.\, \mathsf{let}\ i_1 = \mathsf{cons}(\mathsf{num}_n, i)\ \mathsf{in}\ s_1, i_1 \\
a_{\mathsf{a}} &= \lambda s_1, i \,.\, \mathsf{let}\ h_1, t_1 = \mathsf{dec}(i), h_2, t_2 = \mathsf{dec}(t_1), i_1 = \mathsf{cons}(\mathsf{add}(h_1, h_2), t_2)\ \mathsf{in}\ s_1, i_1 \\
a_{\mathsf{m}} &= \lambda s_1, i \,.\, \mathsf{let}\ h_1, t_1 = \mathsf{dec}(i), h_2, t_2 = \mathsf{dec}(t_1), i_1 = \mathsf{cons}(\mathsf{mul}(h_1, h_2), i_2)\ \mathsf{in}\ s_1, i_1 \\
a_{\mathsf{r}} &= \lambda i \,.\, \mathsf{let}\ h, t = \mathsf{dec}(i)\ \mathsf{in}\ h.
\end{aligned}$$

Definition 7. *A* TSMC with stack *is a tuple* $(\mathbb{C}, \bot, \top, (-)^\infty, \mathsf{cons}, \mathsf{dec})$ *where*

- $(\mathbb{C}, \bot, \top)$ *is a bipointed TSMC,*
- $(-)^\infty : |\mathbb{C}| \to |\mathbb{C}|$ *is a mapping such that* $\mathbf{I}^\infty = \mathbf{I}$ *and* $(A \otimes B)^\infty = A^\infty \otimes B^\infty$,
- $\mathsf{cons}_A : A \otimes A^\infty \to A^\infty$ *and* $\mathsf{dec}_A : A^\infty \to A \otimes A^\infty$ *are $\mathbb{C}$-object indexed families of morphisms such that*

$$\begin{aligned}
&\mathsf{dec}_A \circ \mathsf{cons}_A = \mathrm{id}_{A \otimes A^\infty}, \qquad \mathsf{dec}_{\mathbf{I}} = \mathsf{cons}_{\mathbf{I}} = \mathrm{id}_{\mathbf{I}}, \\
&\mathsf{dec}_A \circ \bot_A = \bot_A \otimes \bot_{A^\infty}, \qquad \top_A \circ \mathsf{cons}_A = \top_A \otimes \top_{A^\infty}, \\
&\mathsf{cons}_{A \otimes B} = (\mathsf{cons}_A \otimes \mathsf{cons}_B) \circ (A \otimes \sigma_{B \otimes A^\infty} \otimes B^\infty), \\
&\mathsf{dec}_{A \otimes B} = (A \otimes \sigma_{A^\infty \otimes B} \otimes B^\infty) \circ (\mathsf{dec}_A \otimes \mathsf{dec}_B).
\end{aligned}$$

A stack-preserving functor *between TSMCs with stack is a traced symmetric strong monoidal functor* $(F, \phi_{\mathbf{I}}, \phi_{A,B})$ *such that it preserves bipoints,* $F(A^\infty) = (FA)^\infty$ *and*

$$F(\mathsf{cons}_A) \circ \phi_{A,A^\infty} = \mathsf{cons}_{FA} \qquad \phi_{A,A^\infty} \circ \mathsf{dec}_{FA} = F(\mathsf{dec}_A).$$

We define $\mathbf{TSMC}^S$ *to be the 2-category of TSMCs with stack, stack-preserving functors and monoidal natural isomorphisms. Its 2-subcategory consisting of traced symmetric strict monoidal functors is denoted by* $\mathbf{TSMC}^S_s$*. We write* $U^S : \mathbf{TSMC}^S \to \mathbf{TSMC}$ *for the canonical forgetful functor.*

Next, U^S can be restricted to a 2-functor $U^S_s : \mathbf{TSMC}^S_s \to \mathbf{TSMC}_s$, and when viewed as an ordinary functor, it has a left adjoint $F^S : \mathbf{TSMC}_s \to \mathbf{TSMC}^S_s$. This left adjoint constructs a syntactic TSMC with stacks from a given category. We omit its detail, but an object of the category $F^S\mathbb{C}$ is an element of the countably infinite coproduct of the monoid $(|\mathbb{C}|, \mathbf{I}, \otimes)$, that is, a sequence $(k_1, C_1)\cdots(k_n, C_n)$ such that $k_i \in \mathbf{N}$, $C_i \in |\mathbb{C}|$ $(1 \le i \le n)$ and $k_i \neq k_j$ if $i \neq j$. The unit $H_\mathbb{C}$ and counit $E_\mathbb{C}$ of the adjunction is defined by (on objects) $H_\mathbb{C}(C) = (0, C)$ and $H_\mathbb{C}(k_1, C_1)\cdots(k_n, C_n) = \bigotimes_{i=1}^n C_i^{\infty(k_i)}$. The category $F^S\mathcal{L}(\mathbb{C})$ admits the strictification operator $\mathbf{Str}^S$ satisfying the condition 4 of Definition 3; on objects it is defined by $\mathbf{Str}^S(F)((k_1, \rho_1)\cdots(k_n, \rho_n)) = \bigotimes_{1\le i\le n, 1\le j\le|\rho_i|}(F\rho_{ij})^{\infty(k_i)}$.

Let $(\mathbb{C}, \bot, \top, (-)^\infty, \mathsf{cons}, \mathsf{dec}) \in \mathbf{TSMC}^S$, which we just write $\mathbb{C}$. We define the tuple $\mathbf{Int}^S(\mathbb{C}) = (\mathbf{Int}(\mathbb{C}), \bot^{\mathbf{Int}(\mathbb{C})}, \top^{\mathbf{Int}(\mathbb{C})}, (-)^{\infty^{\mathbf{Int}(\mathbb{C})}}, \mathsf{cons}^{\mathbf{Int}(\mathbb{C})}, \mathsf{dec}^{\mathbf{Int}(\mathbb{C})})$ by

$$\bot_A^{\mathbf{Int}(\mathbb{C})} = \bot_{A^+} \otimes \top_{A^-}, \qquad \top_A^{\mathbf{Int}(\mathbb{C})} = \top_{A^+} \otimes \bot_{A^-}, \qquad A^{\infty^{\mathbf{Int}(\mathbb{C})}} = ((A^-)^\infty, (A^+)^\infty),$$
$$\mathsf{cons}_A^{\mathbf{Int}(\mathbb{C})} = \mathsf{cons}_{A^+} \otimes \mathsf{dec}_{A^-}, \qquad \mathsf{dec}_A^{\mathbf{Int}(\mathbb{C})} = \mathsf{dec}_{A^+} \otimes \mathsf{cons}_{A^-}.$$

We define $\mathbf{Int}^S$ of 1-cells and 2-cells in $\mathbf{TSMC}^S$ to be **Int** of them. In this way $\mathbf{Int}^S$ becomes a 2-functor. One can check that we can adopt the structure of the pseudo-monad **Int** to make $\mathbf{Int}^S$ a pseudo-monad over $\mathbf{TSMC}^S$. Thus we obtain a pseudo-monad $\mathbf{Int}^S : \mathbf{TSMC}^S \to \mathbf{TSMC}^S$ which is a strict lifting of the pseudo-monad **Int** along U^S.

Proposition 5. *The triple* $\mathcal{E}^S = (F^S, U^S, \mathbf{Int}^S)$ *forms an extension of TSMC.*

Proposition 6. *The tuple* $(\mathcal{S}(\Sigma), \bot, \top, (-)^\infty, \mathsf{cons}, \mathsf{dec})$ *is a TSMC with stack, and is equivalent to* $F^S(\mathcal{L}(\Sigma))$*. Therefore, quasi-SSUR stack ACs are* $\mathcal{E}^S$*-ACs.*

5 Conclusion and Discussion

We presented a categorical framework for capturing various extensions of ACs and their descriptional composition. By setting up appropriate extensions of TSMCs, the descriptional composition of non-deterministic ACs, quasi-SSUR ACs and quasi-SSUR stack ACs are covered by our framework. The framework uniformly guarantees the associativity of the descriptional composition and the closure property of extended ACs under the descriptional composition.

We strongly believe that our framework will contribute to extending the fusion transformation of functions with accumulating parameters. In attribute grammar framework,

the fusion problem is reformulated as the descriptional composition of attribute couplings that represent functions with accumulating parameters. Extending this approach with extra language features is a delicate task (see e.g. [18]), and our categorical framework indicates the direction of the extension of ACs so that the descriptional composition works. For instance, one may consider introducing the map operator to stack ACs. In our framework, this is done by promoting the operator $(-)^{\infty}$ in Section 4.3 to a functor, then form a suitable extension of TSMCs.

We also expect that our framework can provide an alternative account for the existing fusion methods that use circular let bindings to express fusion results as first-order functional programs [22, 19, 13]. An interesting connection between these transformations and the category theory is that, these fusion results, when viewed as morphisms, often have the same pattern as the composition of morphisms in $\mathbf{Int}(\mathbb{C})$; this is also observed in [13]. Through this similarity, our categorical view of the descriptional composition will be helpful to understand these fusion methods, and hopefully provide an equational proof of their correctness.

One possible future work is to implement the descriptional composition based on our categorical framework. The major task will be to define the data structure representing TSMCs and implement the **Int** construction on them. The implementation task breaks down the descriptional composition into fundamental operations on categories and functors, each of which will be easily verifiable.

Acknowledgement The author is grateful to Susumu Nishimura and Craig Pastro for discussions, and Masahito Hasegawa for his encouragement and helpful feedback.

References

1. Miklós Bartha. Linear deterministic attributed transformations. *Acta Cybern.*, 6:125–147, 1983.
2. R. Blackwell, G. M. Kelly, and A. J. Power. Two-dimentional monad theory. *Journal of pure and applied algebra*, 59:1–41, 1989.
3. John Boyland. Conditional attribute grammars. *ACM Trans. Program. Lang. Syst.*, 18(1):73–108, 1996.
4. Laurian M. Chirica and David F. Martin. An order-algebraic definition of Knuthian semantics. *Mathematical Systems Theory*, 13:1–27, 1979.
5. Bruno Courcelle and Pierre Deransart. Proofs of partial correctness for attribute grammars with applications to recursive procedures and logic programming. *Inf. Comput.*, 78(1):1–55, 1988.
6. Harald Ganzinger. Increasing modularity and language-independency in automatically generated compilers. *Sci. Comput. Program.*, 3(3):223–278, 1983.
7. Harald Ganzinger and Robert Giegerich. Attribute coupled grammars. In *SIGPLAN Symposium on Compiler Construction '84*, pages 157–170. ACM, 1984.
8. Robert Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Inf.*, 25(4):355–423, 1988.
9. Masahito Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of* let *and* letrec. Springer-Verlag, 1999.
10. Thomas Johnsson. Attribute grammars as a functional programming paradigm. In Gilles Kahn, editor, *FPCA*, volume 274 of *Lecture Notes in Computer Science*, pages 154–173. Springer, 1987.

11. Andre Joyal, Ross Street, and Dominc Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.
12. Shin-ya Katsumata. Attribute grammars and categorical semantics. In Luca Aceto, Ivan Damgrd, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 271–282. Springer, 2008.
13. Shin-ya Katsumata and Susumu Nishimura. Algebraic fusion of functions with an accumulating parameter and its improvement. In John H. Reppy and Julia L. Lawall, editors, *ICFP*, pages 227–238. ACM, 2006.
14. Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.
15. Donald E. Knuth. Correction: Semantics of context-free languages. *Mathematical Systems Theory*, 5(1):95–96, 1971.
16. Anders Kock. Strong functors and monoidal monads. *Archiv der Math.*, 23(1):113–120, 1972.
17. Saunders MacLane. *Categories for the Working Mathematician (Second Edition)*, volume 5 of *Graduate Texts in Mathematics*. Springer, 1998.
18. Keisuke Nakano. Composing stack-attributed tree transducers. *Theory Comput. Syst.*, 44(1):1–38, 2009.
19. Susumu Nishimura. Deforesting in accumulating parameters via type-directed transformations. In *APLAS '02*, pages 145–159, 2002.
20. Susumu Nishimura and Keisuke Nakano. XML stream transformer generation through program composition and dependency analysis. *Sci. Comput. Program.*, 54(2-3):257–290, 2005.
21. Miki Tanaka and John Power. Pseudo-distributive laws and axiomatics for variable binding. *Higher-Order and Symbolic Computation*, 19(2-3):305–337, 2006.
22. Janis Voigtländer. Using circular programs to deforest in accumulating parameters. *Higher-Order and Symbolic Computation*, 17(1-2):129–163, 2004.

A Proof of Theorem 2

1) Obvious. 2) First, we introduce an auxiliary binary operator $©'$. For functors $\mathcal{A} \in \mathbf{TSMC}'_s(F\mathcal{L}(\Delta), \mathbf{Int}'F\mathcal{L}(\Sigma))$ and $\mathcal{B} \in \mathbf{TSMC}'_s(F\mathcal{L}(\Sigma), \mathbf{Int}'(\mathbb{C}))$, we define $\mathcal{B}\ ©'\ \mathcal{A}$ to be $\mathbf{Str}'((\mathcal{B})^\flat \circ \mathcal{A})$. It is associative up to an isomorphism:

$$C\ ©'\ (\mathcal{B}\ ©'\ \mathcal{A}) = \mathbf{Str}'((C)^\flat \circ (\mathcal{B})^\flat \circ \mathcal{A}) \cong \mathbf{Str}'(((C)^\flat \circ \mathcal{B})^\flat \circ \mathcal{A}) \cong (C\ ©'\ \mathcal{B})\ ©'\ \mathcal{A}.$$

Let $\mathcal{A} \in \mathbf{TSMC}_s(\mathcal{L}(\Delta), \mathbf{Int}UF\mathcal{L}(\Sigma))$ and $\mathcal{B} \in \mathbf{TSMC}_s(\mathcal{L}(\Sigma), \mathbf{Int}U\mathbb{C})$. Then we have

$$\mathcal{B}\ ©\ \mathcal{A} = \mathbf{Str}(U(\underline{\mathcal{B}})^\flat) \circ \mathcal{A}) = \mathbf{Str}(\overline{(\underline{\mathcal{B}})^\flat \circ \underline{\mathcal{A}}}) = \overline{\mathbf{Str}'((\underline{\mathcal{B}})^\flat \circ \underline{\mathcal{A}})} = \overline{(\underline{\mathcal{B}}\ ©'\ \underline{\mathcal{A}})}.$$

The bijection $\overline{(-)} : |\mathbf{TSMC}'_s(F\mathbb{C}, \mathbb{D})| \to |\mathbf{TSMC}_s(\mathbb{C}, U\mathbb{D})|$ extends to an ordinary functor $\mathbf{TSMC}'_s(F\mathbb{C}, \mathbb{D}) \to \mathbf{TSMC}_s(\mathbb{C}, U\mathbb{D})$, because it is defined as $\overline{F} = UF \circ H_{\mathbb{C}}$. Therefore if F and G are naturally isomorphic, so are $\overline{F}$ and $\overline{G}$. Then we have

$$C\ ©\ (\mathcal{B}\ ©\ \mathcal{A}) = \overline{\underline{C}\ ©'\ (\underline{\mathcal{B}}\ ©'\ \underline{\mathcal{A}})} \cong \overline{(\underline{C}\ ©'\ \underline{\mathcal{B}})\ ©'\ \underline{\mathcal{A}}} = C\ ©\ (\mathcal{B}\ ©\ \mathcal{A}).$$

3) Note that $N(T\mathcal{A}(H(f))) = \mathcal{A}(f)$. Since N is faithful, it is sufficient to show $N \circ T(\mathcal{B}\ ©\ \mathcal{A}) \circ H(f) = N \circ T\mathcal{B} \circ T\mathcal{A} \circ H(f)$.

$$\begin{aligned} N(T(\mathcal{B}\ ©\ \mathcal{A})(H(f))) &= \mathbf{Str}((U\underline{\mathcal{B}})^\# \circ \mathcal{A})(f) = (U\underline{\mathcal{B}})^\#(N(T\mathcal{A}(H(f)))) \\ &= (U\underline{\mathcal{B}})(T\mathcal{A}(H(f))) = N(T\mathcal{B}(T\mathcal{A}(H(f)))). \end{aligned}$$

B Traced Symmetric Monoidal Categories and the Int Construction

Traced Symmetric Monoidal Category We recall the concept of trace operator [11]. A *trace operator* on a symmetric strict monoidal category $(\mathbb{C}, \mathbf{I}, \otimes, \sigma)$ is a mapping $\mathrm{tr}^A_{B,C} : \mathbb{C}(B \otimes A, C \otimes A) \to \mathbb{C}(B, C)$ satisfying the following equations.

$$\begin{array}{lr} \text{(Naturality)} & h \circ \mathrm{tr}^A_{B,C}(f) \circ g = \mathrm{tr}^A_{B',C'}((h \otimes A) \circ f \circ (g \otimes A)) \\ \text{(Dinaturality)} & \mathrm{tr}^A_{B,C}((C \otimes g) \circ f) = \mathrm{tr}^{A'}_{B,C}(f \circ (B \otimes g)) \\ \text{(Vanishing I)} & \mathrm{tr}^{\mathbf{I}}_{A,B}(f) = f \\ \text{(Vanishing II)} & \mathrm{tr}^{A\otimes B}_{C,D}(g) = \mathrm{tr}^A_{C,D}(\mathrm{tr}^B_{C\otimes A, D\otimes A}(g)) \\ \text{(Superposing)} & \mathrm{tr}^A_{B\otimes C, B\otimes D}(B \otimes f) = B \otimes \mathrm{tr}^A_{C,D} f \\ \text{(Yanking)} & \mathrm{tr}^A_{A,A}(\sigma_{A,A}) = \mathrm{id}. \end{array}$$

We simplify the superposing axiom in [11] using naturality and dinaturality [9]. A *traced symmetric strict monoidal category* (TSMC) is a pair of a symmetric strict monoidal category and a trace operator on it. We say that a symmetric strong monoidal functor $(F, \phi_{\mathbf{I}}, \phi_{A,B}) : \mathbb{C} \to \mathbb{D}$ between TSMCs $\mathbb{C}, \mathbb{D}$ is *traced* if it satisfies $F(\mathrm{tr}^C_{A,B}(f)) = \mathrm{tr}^{FC}_{FA,FB}(\phi^{-1}_{B,C} \circ Ff \circ \phi_{A,C})$.

The **Int** *Construction* Joyal, Street and Verity showed that the forgetful functor from the 2-category of tortile monoidal categories to that of traced (braided) monoidal categories has a left biadjoint, which they called **Int** [11]. This biadjunction can be restricted to the one between **TSMC** and the 2-category of strict compact closed categories, strong monoidal functors and monoidal natural isomorphisms. Like the usual construction of monads from adjunctions, we obtain a pseudo-monad over **TSMC**, which we also write **Int**. Below we give an explicit definition of this pseudo-monad **Int**.

Let $\mathbb{C} \in \mathbf{TSMC}$. We define the TSMC $\mathbf{Int}(\mathbb{C})$ by the following data. An object is a pair (A^+, A^-) of $\mathbb{C}$-objects. Below, when A is declared as an $\mathbf{Int}(\mathbb{C})$-object, by A^+ and A^- we mean its first and second component. A morphism from A to B is a $\mathbb{C}$-morphism $f : A^+ \otimes B^- \to B^+ \otimes A^-$. The identity is defined by $\mathrm{id}_A^{\mathbf{Int}(\mathbb{C})} = \mathrm{id}_{A^+ \otimes A^-}$, and the composition of $f : A \to B$ and $g : B \to C$ is defined by $g \circ^{\mathbf{Int}(\mathbb{C})} f = \mathrm{tr}^{B^-}_{A^+ \otimes C^-, C^+ \otimes A^-}(h)$ where $h = (C^+ \otimes \sigma_{B^-,A^-}) \circ (g \otimes A^-) \circ (B^+ \otimes \sigma_{A^-,C^-}) \circ (f \otimes C^-) \circ (A^+ \otimes \sigma_{C^-,B^-})$. The (strict) tensor product in $\mathbf{Int}(\mathbb{C})$ is given by $A \otimes^{\mathbf{Int}(\mathbb{C})} B = (A^+ \otimes B^+, A^- \otimes B^-)$ and $f \otimes^{\mathbf{Int}(\mathbb{C})} g = (B^+ \otimes \sigma_{A^-,D^+} \otimes C^-) \circ (f \otimes g) \circ (A^+ \otimes \sigma_{C^+,B^-} \otimes D^-)$. The unit object of this tensor product is given by $(\mathbf{I}, \mathbf{I})$. The symmetry morphism is $\sigma^{\mathbf{Int}(\mathbb{C})}_{A,B} = \sigma_{A^+,B^+} \otimes \sigma_{A^-,B^-}$. We give the trace operator with respect to the above symmetric monoidal structure by $(\mathrm{tr}^{\mathbf{Int}(\mathbb{C})})^C_{A,B}(f) = \mathrm{tr}^{C^+ \otimes C^-}_{A^+ \otimes B^-, B^+ \otimes A^-}((B^+ \otimes \sigma_{C^+,A^-} \otimes C^-) \circ f \circ (A^+ \otimes \sigma_{B^-,C^+} \otimes C^-))$.

We next give a functor $\mathbf{Int}_{\mathbb{C},\mathbb{D}} : \mathbf{TSMC}(\mathbb{C}, \mathbb{D}) \to \mathbf{TSMC}(\mathbf{Int}(\mathbb{C}), \mathbf{Int}(\mathbb{D}))$, which we simply write by **Int**. We define **Int** of a traced symmetric strong monoidal functor $(F, \phi_{\mathbf{I}}, \phi_{A,B}) : \mathbb{C} \to \mathbb{D}$ to be the tuple $(\mathbf{Int}(F), \phi_{\mathbf{I}}^{\mathbf{Int}(F)}, \phi_{A,B}^{\mathbf{Int}(F)}) : \mathbf{Int}(\mathbb{C}) \to \mathbf{Int}(\mathbb{D})$ defined by $\mathbf{Int}(F)(A) = (FA^+, FA^-)$, $\mathbf{Int}(F)(f) = \phi^{-1}_{B^+,B^-} \otimes Ff \otimes \phi_{A^+,A^-}$, $\phi_{\mathbf{I}}^{\mathbf{Int}(F)} = \phi_{\mathbf{I}} \otimes \phi_{\mathbf{I}}^{-1}$ and $\phi_{A,B}^{\mathbf{Int}(F)} = \phi_{A^+,B^+} \otimes \phi^{-1}_{A^-,B^-}$. For a monoidal natural isomorphism $\alpha : F \to G$, we define $\mathbf{Int}(\alpha) : \mathbf{Int}(F) \to \mathbf{Int}(G)$ by $\mathbf{Int}(\alpha)_A = \alpha_{A^+} \otimes \alpha^{-1}_{A^-}$. These data determine a 2-endofunctor $\mathbf{Int} : \mathbf{TSMC} \to \mathbf{TSMC}$.

Let $\mathbb{C} \in \mathbf{TSMC}$. A calculation shows that the homset $\mathbf{Int}^2(\mathbb{C})(A, B)$ is identical to $\mathbb{C}(A^{++} \otimes B^{-+} \otimes B^{+-} \otimes A^{--}, B^{++} \otimes A^{-+} \otimes A^{+-} \otimes B^{--})$; thus we manipulate morphisms in $\mathbf{Int}^2(\mathbb{C})$ as $\mathbb{C}$-morphisms. The unit of the pseudo-monad **Int** is the traced symmetric strict monoidal functor $N_{\mathbb{C}} : \mathbb{C} \to \mathbf{Int}(\mathbb{C})$ defined by $N_{\mathbb{C}}(A) = (\mathbf{I}, A)$ and $N_{\mathbb{C}}(f) = f$, while the multiplication of **Int** is the traced symmetric strong monoidal functor $M_{\mathbb{C}} : \mathbf{Int}^2(\mathbb{C}) \to \mathbf{Int}(\mathbb{C})$ defined by $M_{\mathbb{C}}(A) = (A^{--} \otimes A^{++}, A^{-+} \otimes A^{+-})$ and $M_{\mathbb{C}}(f) = r^{-1}_{B,A} \circ f \circ r_{A,B}$, where $r_{A,B}$ is the symmetry morphism $\sigma_{A^{--},A^{++} \otimes B^{-+} \otimes B^{+-}}$ in $\mathbb{C}$. We also define monoidal natural isomorphisms $n_F : \mathbf{Int}(F) \circ N_{\mathbb{C}} \to N_{\mathbb{D}} \circ F$ and $m_F : \mathbf{Int}^2(F) \circ M_{\mathbb{C}} \to M_{\mathbb{D}} \circ \mathbf{Int}(F)$ for $(F, \phi_{\mathbf{I}}, \phi_{A,B}) \in \mathbf{TSMC}(\mathbb{C}, \mathbb{D})$ by $(n_F)_A = \mathrm{id}_{FA} \otimes \phi_{\mathbf{I}}$ and $(m_F)_A = \phi^{-1}_{A^{--},A^{++}} \otimes \phi_{A^{-+},A^{+-}}$. These data determine pseudo-natural transformations $(N, n) : \mathrm{Id} \to \mathbf{Int}$ and $(M, m) : \mathbf{Int}^2 \to \mathbf{Int}$.

Finally, we define modifications $\lambda_{\mathbb{C}} : M_{\mathbb{C}} \circ \mathbf{Int}(N_{\mathbb{C}}) \to \mathrm{Id}_{\mathbf{Int}(\mathbb{C})}$ and $\rho_{\mathbb{C}} : M_{\mathbb{C}} \circ N_{\mathbf{Int}(\mathbb{C})} \to \mathrm{Id}_{\mathbf{Int}(\mathbb{C})}$ to be identities, and $\tau_{\mathbb{C}} : M_{\mathbb{C}} \circ \mathbf{Int}(M_{\mathbb{C}}) \to M_{\mathbb{C}} \circ M_{\mathbf{Int}(\mathbb{C})}$ by $(\tau_{\mathbb{C}})_A = (\sigma_{A^{--+},A^{-+-} \otimes A^{+--}} \otimes A^{+++}) \otimes (\sigma^{-1}_{A^{---},A^{-++} \otimes A^{+-+}} \otimes A^{++-})$.

Theorem 3. *The tuple* $(\mathbf{Int}, (N, n), (M, m), \tau, \lambda, \rho)$ *forms a pseudo-monad on* **TSMC**.