

Categorical Descriptive Composition

Shin-ya Katsumata

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

Abstract. The descriptive 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 descriptive composition based on traced symmetric monoidal categories and the **Int** construction by Joyal et al. We demonstrate that this framework can handle the descriptive composition of SSUR-ACs, nondeterministic SSUR-ACs, quasi-SSUR ACs and stack ACs.

1 Introduction

Descriptive composition [5–7] is a method to fuse two term transformation algorithms described by attribute couplings (ACs; attribute grammars (AGs) over terms) into one. The AC yielded by the descriptive 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 descriptive composition was first introduced as an optimisation method for compilers described by ACs [6]. Around the same time Bartha introduced a similar composition method for linear attributed tree transformations [1]. Later, it was realised that the AG framework can be used to represent functional programs with accumulating parameters [9], and the descriptive composition inspired various fusion transformations of such functional programs [17, 19].

The descriptive composition was first given for the ACs consisting only of term constructors [6]. Later, extensions of the ACs have been studied in [16, 18, 2]. In [2], Boyland and considered an extension of ACs with conditional expressions. In [16], Nakano introduce stacks to ACs so that complex parsing functions can be expressed. In each work, the descriptive composition was also considered to these extended ACs. In general, the descriptive composition is sensitive on the language describing ACs, and it tends to get involved when expressive power of the language increases.

The question we address here is to find a mathematical framework that can uniformly treat these extension of ACs and the descriptive composition. 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 AGs, ACs and the descriptive composition is that every AG determines a traced symmetric monoidal functor $F : \mathcal{L}(\Sigma) \rightarrow \mathbf{Int}(\mathbb{C})$ where $\mathcal{L}(\Sigma)$ is a free TSMC over a signature Σ , and every AC satisfying *syntactic single use condition* (SSUR), which is the essential condition for the descriptive composition

to work, determines a traced symmetric monoidal functor $G : \mathcal{L}(\Sigma) \rightarrow \mathbf{Int}(\mathcal{L}(\Delta))$. In general, composing ACs is an intricate task, but with this functorial presentation, the descriptive composition becomes the composition of functors, and it is trivially associative. 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 [16].

We note that in this paper we adopt mathematical structures that admit circular terms / recursive computation (namely TSMCs), then discuss AGs and the descriptive composition over them. Hence the issue of well-definiteness of the meaning does not occur; every AG assigns a meaning to a given term.

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

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

For a type $\sigma \in \Sigma$ and sequence of Σ -types ρ_1, \dots, ρ_n , by $T_\Sigma^\sigma(\rho_1, \dots, \rho_n)$ we mean the set of open Σ -terms that may contain some variables x_i of type ρ_i ($1 \leq i \leq n$). We then extend this notation for a sequence $\sigma_1, \dots, \sigma_n$ of Σ -types by $T_\Sigma^{\sigma_1, \dots, \sigma_n}(\rho_1, \dots, \rho_n) = T_\Sigma^{\sigma_1}(\rho_1, \dots, \rho_n) \times \dots \times T_\Sigma^{\sigma_n}(\rho_1, \dots, \rho_n)$. By Σ^+ we mean that Σ contains a special type $\#$. We assume that the readers are familiar with the concept of symmetric monoidal categories [15].

2 Classical Attribute Couplings and Descriptive 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 sets of domains of inherited and synthesised attributes, and for each operator $o \in \Delta^{\rho_1 \cdots \rho_n \rightarrow \rho}$, a_o is a function called the *attribute calculation rule*¹:

$$a_o : S\rho_1 \times \dots \times S\rho_n \times I\rho \rightarrow S\rho \times I\rho_1 \times \dots \times I\rho_n. \quad (1)$$

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 [12, 13] 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 $T_\Sigma^{\sigma_1, \dots, \sigma_n}$ of tuples of Σ -terms for some $\sigma_1, \dots, \sigma_n \in \Sigma$, and 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

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

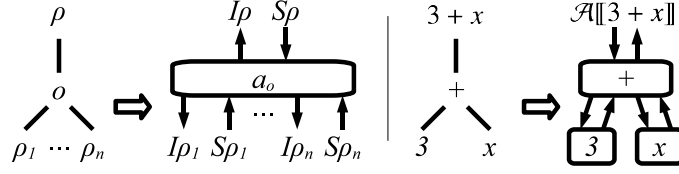


Fig. 1. Attribute Grammars

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 $o \in \Delta^{\rho_1, \dots, \rho_n \rightarrow \rho}$, a_o is a tuple of (open) Σ -terms:

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

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 $T\mathcal{A} : T_{\Delta^+}^{\#} \rightarrow T_{\Sigma^+}^{\#}$, 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 $\mathcal{B} \circledast \mathcal{A}$ such that $T(\mathcal{B} \circledast \mathcal{A}) = T\mathcal{B} \circ T\mathcal{A}$. In general such AC may not exist, but when \mathcal{A} satisfies a condition called *syntactic single-use restriction (SSUR)*, we can build $\mathcal{B} \circledast \mathcal{A}$ by the *descriptive 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,

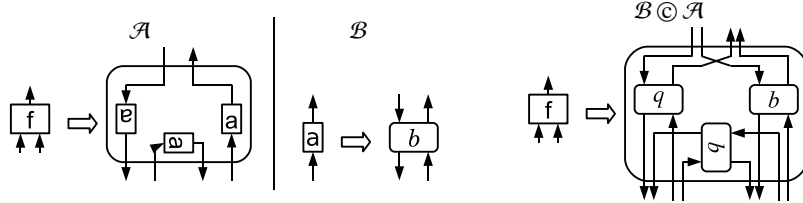


Fig. 2. The descriptive composition

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 constructors by the computation unit assigned to that constructor 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 descriptive composition $\mathcal{B} \circledast \mathcal{A}$.

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

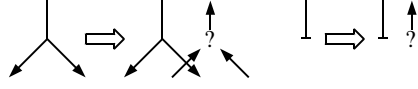


Fig. 3. How to Make Branches and Terminals Bidirectional?

$$\begin{array}{c}
 \frac{\rho \in \Sigma}{x : \rho \vdash_E x : \rho} \quad \frac{o \in \Sigma^{\rho_1 \cdots \rho_n \rightarrow \rho}}{x_1 : \rho_1, \dots, x_n : \rho_n \vdash_E o(x_1, \dots, x_n) : \rho} \\
 \hline
 \frac{\Gamma_1 \vdash_E M_1 : \rho_1 \quad \dots \quad \Gamma_l \vdash_E M_l : \rho_l \quad U = D}{\vdash_M \lambda x . \text{let } y_1 = M_1, \dots, y_l = M_l \text{ in } z : \sigma_1, \dots, \sigma_n \rightarrow \tau_1, \dots, \tau_m} \\
 \\
 (\lambda x . \text{let } D \text{ in } z) = (\lambda x . \text{let } \pi(D) \text{ in } z) \\
 (\lambda x . \text{let } v = w, D \text{ in } z) = (\lambda x . \text{let } D[w/v] \text{ in } z[w/v]) \quad (v \neq w)
 \end{array}$$

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

each attribute calculation rule assigned by \mathcal{A} should use each variable exactly once, and an AC satisfying this linearity condition is called 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:

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

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

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

such that x_i, y_j, z_k and variables in $\Gamma_1, \dots, \Gamma_n$ are different from each other. The leading λ of expressions is a formal binder for x_1, \dots, 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 $\mathcal{L}(\Sigma_{\text{tree}})$ -expressions:

$$\begin{aligned}
 \vdash_M \lambda x . \text{let } y = \mathbf{N}(l, x), l = \mathbf{L} \text{ in } y : * \rightarrow * \\
 \vdash_M \lambda x, y, z . \text{let } w = \mathbf{N}(x, w), l = \mathbf{L}, v = \mathbf{N}(l, z) \text{ in } y, v : *** \rightarrow **. \quad (2)
 \end{aligned}$$

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.

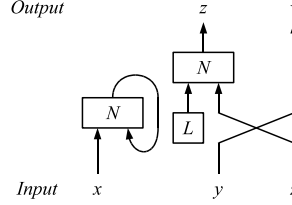


Fig. 5. Network of a $\mathcal{L}(\Sigma)$ -expression

We can draw a network for each $\mathcal{L}(\Sigma)$ -expression. Let us consider drawing the network of (2). We put the input variables x, y, z on the bottom and output variables z, v on the top, then draw a node for each occurrence of an operator in the declaration part of the expression. We then draw edges between vertexes of the nodes according to the usage of variables. Due to the linearity constraint, there is no branching edges in the network.

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

An AC $(I, S, a) \in \mathbf{AC}(\Delta, \Sigma)$ satisfies SSUR if each attribute calculation rule satisfies the following condition. We let l be the length of $S\rho, I\rho_1, \dots, I\rho_n$, 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 $S\rho_1, \dots, S\rho_n, I\rho$ is a permutation of the concatenation $\Gamma_1, \dots, \Gamma_l$. We then ask the attribute calculation rule to be in the following set:

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

and, moreover, in each 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 $S\rho_1, \dots, S\rho_n, I\rho \rightarrow S\rho, I\rho_1, \dots, I\rho_n$. 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 $o \in \Delta^{\rho_1, \dots, \rho_n \rightarrow \rho}$, a_o is an $\mathcal{L}(\Sigma)$ -expression of type $a_o : S\rho_1, \dots, S\rho_n, I\rho \rightarrow S\rho, I\rho_1, \dots, I\rho_n$. We write $\mathbf{SSUR-AC}(\Delta, \Sigma)$ 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 circular expressions (like Figure 5) 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 $\rho_1 \cdots \rho_n$ of types in Σ as an object, and an equivalence class of expressions of type $\rho \rightarrow \sigma$ as a morphism from ρ to σ . The composition is defined by

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

here we assume that every bound variable in the above expression is distinct from each other. The category $\mathcal{L}(\Sigma)$ has an evident strict symmetric 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:

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

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

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

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

Proposition 1. *The category $\mathcal{L}(\Sigma)$ together with the above-mentioned strict symmetric monoidal structure and the trace operator form a traced strict symmetric monoidal category (see Appendix A for the definition).*

We call a strict symmetric monoidal functor between traced strict SMCs *traced* if it preserves the trace operator in an evident way (see Appendix A). We write \mathbf{TSMC}_s for the category of traced strict symmetric monoidal small categories and traced strict symmetric monoidal functors between them.

3.2 Monoidal Attribute Grammar

We “categorify” classical attribute grammars. We replace sets by objects in some $\mathbb{C} \in \mathbf{TSMC}_s$, each attribute calculation rule (1) by a \mathbb{C} -morphism, and Cartesian products with tensor products. We then obtain the concept of *monoidal attribute grammar* [11].

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

Some instances of MAGs are studied in [11]; in the category $\omega\mathbf{CPPO}$ of pointed CPOs and ω -continuous functions, monoidal attribute grammars are equivalent to Chirica and Martin’s K-systems [3]. The category \mathbf{Rel} of sets and relations has traced biproducts [10], 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* [4]. In addition to this, by comparing Definition 1 and 2, we conclude that SSUR-ACs are MAGs in $\mathcal{L}(\Sigma)$.

Proposition 2. $\text{SSUR-AC}(\mathcal{A}, \Sigma) = \text{MAG}(\mathcal{A}, \mathcal{L}(\Sigma))$.

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

Below we give two equivalent concepts of MAGs: one is algebras in the categories obtained by Joyal et al.'s **Int** construction, and the other is traced strict symmetric monoidal functors of type $\mathcal{L}(\Sigma) \rightarrow \mathbf{Int}(\mathbb{C})$, where $\mathbb{C} \in \mathbf{TSMC}_s$.

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

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

The category $\mathbf{Int}(\mathbb{C})$ is symmetric monoidal:

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

The category $\mathbf{Int}(\mathbb{C})$ has a *compact closed structure* [10], which yields the *canonical trace operator* with respect to the above symmetric monoidal structure (see Appendix A). Thus the mapping $\mathbb{C} \mapsto \mathbf{Int}(\mathbb{C})$ extends to an endofunctor over \mathbf{TSMC}_s , and moreover, to a monad (\mathbf{Int}, N, M) over \mathbf{TSMC}_s .²

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 $a_o : A\rho_1 \otimes \cdots \otimes A\rho_n \rightarrow A\rho$ for each operator $o \in \Sigma^{\rho_1 \cdots \rho_n \rightarrow \rho}$. We write $\mathbf{Alg}_{\Sigma}(\mathbb{C})$ for the set of Σ -algebras in \mathbb{C} . The concept of Σ -algebras has another presentation: there is a natural bijection between \mathcal{A} -algebras in \mathbb{C} and traced strict symmetric monoidal functors from $\mathcal{L}(\mathcal{A})$ to \mathbb{C} :

$$\mathbf{Alg}_{\mathcal{A}}(\mathbb{C}) \simeq \mathbf{TSMC}_s(\mathcal{L}(\mathcal{A}), \mathbb{C}). \quad (4)$$

Let (I, S, a) be a MAG for \mathcal{A} in $\mathbb{C} \in \mathbf{TSMC}_s$. We define $A\rho$ to be the pair $(I\rho, S\rho)$ of \mathbb{C} -objects (note that it is an object in $\mathbf{Int}(\mathbb{C})$). Then for each operator $o \in \Sigma^{\rho_1 \cdots \rho_n \rightarrow \rho}$, the \mathbb{C} -morphism a_o is an $\mathbf{Int}(\mathbb{C})$ -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 \mathcal{A} -algebra (A, a) in $\mathbf{Int}(\mathbb{C})$. Conversely, every \mathcal{A} -algebra in $\mathbf{Int}(\mathbb{C})$ immediately determines a MAG in \mathbb{C} . To summarise,

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

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

² The strictness condition is essential for **Int** to be a monad over \mathbf{TSMC}_s .

These three equivalent forms have different advantages. The first form is the actual data we write when concretely defining attribute grammars. The second form is used to explain the initial algebra semantics of attribute grammars. In the third form, we can easily discuss the composition of attribute grammars. We transparently switch the representation of monoidal attribute grammars below.

We now define the transformation of terms induced by ACs. Let $\mathcal{A} = (I, S, a) \in \mathbf{SSUR}\text{-AC}(\mathcal{A}^+, \Sigma^+)$. The transformation of \mathcal{A} -terms induced by \mathcal{A} , written $T\mathcal{A}$, is the function sending $f \in \mathcal{L}(\mathcal{A}^+)(\epsilon, \#)$ to the unique $g \in \mathcal{L}(\Sigma^+)(\epsilon, \#)$ such that $N_{\mathcal{L}(\Sigma^+)} = \mathcal{A}(f)$. Such g always exists as N is full and faithful [10].

4 Descriptive Composition

We begin with a categorical formulation of the descriptive composition of SSUR-ACs. Let $\mathcal{A} \in \mathbf{SSUR}\text{-AC}(\mathcal{A}, \Sigma)$ and $\mathcal{B} \in \mathbf{MAG}(\Sigma, \mathbb{C})$, regarded as functors. We define their categorical descriptive composition $\mathcal{B} \odot \mathcal{A}$ to be the composite $\mathcal{B}^\# \circ \mathcal{A}$:

$$\begin{array}{ccc} \mathcal{L}(\mathcal{A}) & \xrightarrow{\mathcal{A}} & \mathbf{Int}(\mathcal{L}(\Sigma)) \\ & & \uparrow N \\ & & \mathcal{L}(\Sigma) \xrightarrow{\mathcal{B}} \mathbf{Int}(\mathbb{C}) \\ & & \searrow \mathcal{B}^\# \end{array}$$

where $(-)^\#$ is the Kleisli lifting of the monad \mathbf{Int} . The composition is a MAG for Σ in \mathbb{C} . The following theorem is almost immediate.

Theorem 1. 1. For any $\mathcal{A} \in \mathbf{SSUR}\text{-AC}(\mathcal{A}, \Sigma)$, $\mathcal{B} \in \mathbf{SSUR}\text{-AC}(\Sigma, \mathbb{E})$ and any MAG $\mathcal{C} \in \mathbf{MAG}(\mathbb{E}, \mathbb{C})$ we have $(\mathcal{C} \odot \mathcal{B}) \odot \mathcal{A} = \mathcal{C} \odot (\mathcal{B} \odot \mathcal{A})$.
 2. SSUR-ACs are closed under descriptive composition.
 3. For any $\mathcal{A} \in \mathbf{SSUR}\text{-AC}(\mathcal{A}^+, \Sigma^+)$ and $\mathcal{B} \in \mathbf{SSUR}\text{-AC}(\Sigma^+, \mathbb{E}^+)$, we have $T\mathcal{B} \circ T\mathcal{A} = T(\mathcal{B} \odot \mathcal{A})$.

We extend this formulation of the descriptive composition to a more general setting where traced strict SMCs are equipped with “extra structures”, such as nondeterminism, undefined values, stacks, etc. We assume that such traced strict SMCs with extra structures form a category \mathbf{TSMC}'_s , and it comes with an adjunction $(F \dashv U, \eta, \epsilon) : \mathbf{TSMC}'_s \rightarrow \mathbf{TSMC}_s$. We also assume that there is a lifting \mathbf{Int}' of \mathbf{Int} across U : the lifting is given by a monad $(\mathbf{Int}', N', M') : \mathbf{TSMC}'_s \rightarrow \mathbf{TSMC}'_s$:

$$\begin{array}{ccc} \mathbf{TSMC}'_s & \xrightarrow{\mathbf{Int}'} & \mathbf{TSMC}'_s \\ \uparrow F \downarrow U & & \uparrow F \downarrow U \\ \mathbf{TSMC}_s & \xrightarrow{\mathbf{Int}} & \mathbf{TSMC}_s \end{array}$$

such that for every $\mathbb{C}' \in \mathbf{TSMC}'_s$, $U(N'_{\mathbb{C}'}) = N_{U\mathbb{C}'}$ and $U(M'_{\mathbb{C}'}) = M_{U\mathbb{C}'}$. Below we call the above situation an *extension of TSMCs*, and express it by a tuple $\mathcal{E} = (F \dashv U, \eta, \epsilon, \mathbf{Int}', N', M')$; we may omit writing η, ϵ, N', M' when they are not referred in the context.

Definition 3. Let $\mathcal{E} = (F \dashv U, \mathbf{Int}')$ be an extension of TSMCs. An \mathcal{E} -MAG for Δ in $\mathbb{C} \in \mathbf{TSMC}'_s$ is just a MAG for Δ in UC. An \mathcal{E} -AC from Δ to Σ is just a MAG for Δ in $UF\mathcal{L}(\Delta)$. We write $\mathcal{E}\text{-MAG}(\Delta, \mathbb{C})$ and $\mathcal{E}\text{-AC}(\Delta, \Sigma)$ for the sets of \mathcal{E} -MAGs and \mathcal{E} -ACs, respectively.

From (5) and (6), \mathcal{E} -MAGs and \mathcal{E} -ACs correspond to traced strict symmetric monoidal functors:

$$\mathcal{E}\text{-MAG}(\Delta, \mathbb{C}) \simeq \mathbf{TSMC}_s(\mathcal{L}(\Delta), \mathbf{Int}(UC)), \quad (7)$$

$$\mathcal{E}\text{-AC}(\Delta, \Sigma) \simeq \mathbf{TSMC}_s(\mathcal{L}(\Delta), \mathbf{Int}(UF\mathcal{L}(\Sigma))). \quad (8)$$

Let $\mathcal{A} \in \mathcal{E}\text{-AC}(\Delta, \Sigma)$ and $\mathcal{B} \in \mathcal{E}\text{-MAG}(\Sigma, \mathbb{C})$, regarded as functors. We define their descriptonal composition $\mathcal{B} \odot \mathcal{A}$ to be the composite $\overline{\mathcal{B}}^\# \circ \mathcal{A}$:

$$\begin{array}{ccc}
 \mathcal{L}(\Delta) & \xrightarrow{\mathcal{A}} & \mathbf{Int}(UF\mathcal{L}(\Sigma)) \\
 & & \uparrow N_{UF\mathcal{L}(\Delta)} \\
 & & UF\mathcal{L}(\Sigma) \\
 & & \uparrow \eta_{\mathcal{L}(\Sigma)} \\
 \mathcal{L}(\Sigma) & \xrightarrow{\mathcal{B}} & \mathbf{Int}(UC) = U\mathbf{Int}'(\mathbb{C})
 \end{array}
 \begin{array}{l}
 \\
 \\
 \searrow \overline{\mathcal{B}}^\# \\
 \searrow \overline{\mathcal{B}}
 \end{array}$$

where $\overline{\mathcal{B}} = U\epsilon_{\mathbf{Int}'(\mathbb{C})} \circ UF\mathcal{B}$.

For signatures Δ, Σ containing the special type $\#$, we assume that every $\mathcal{A} \in \mathcal{E}\text{-AC}(\Delta, \Sigma)$, regarded as a functor, satisfies $\mathcal{A}(\#) = N_{UF\mathcal{L}(\Sigma)}(\eta_{\mathcal{L}(\Sigma)}(\#))$. The transformation of Δ -terms induced by \mathcal{A} , written $T\mathcal{A}$, is the function sending $f \in \mathcal{L}(\Delta)(\epsilon, \#)$ to the unique $g \in UF\mathcal{L}(\Sigma)(\mathbf{I}, \eta_{\mathcal{L}(\Sigma)}(\#))$ such that $N_{UF\mathcal{L}(\Sigma)}(g) = \mathcal{A}(f)$. Such unique g always exists as N is full and faithful [10]. We note that $T\mathcal{A}$ yields morphisms in $UF\mathcal{L}(\Delta)$ because the attribute calculation rules in \mathcal{A} may construct some Δ -terms containing the extra structures provided by $F\mathcal{L}(\Delta)$.

Theorem 2. Let \mathcal{E} be an extension of TSMCs.

1. For any $\mathcal{A} \in \mathcal{E}\text{-AC}(\Delta, \Sigma)$, $\mathcal{B} \in \mathcal{E}\text{-MAG}(\Sigma, \mathbb{E})$ and any MAG $C \in \mathcal{E}\text{-MAG}(\mathbb{E}, \mathbb{C})$ we have $(C \odot \mathcal{B}) \odot \mathcal{A} = C \odot (\mathcal{B} \odot \mathcal{A})$.
2. \mathcal{E} -ACs are closed under descriptonal composition.
3. For any $\mathcal{A} \in \mathcal{E}\text{-AC}(\Delta^+, \Sigma^+)$ and $\mathcal{B} \in \mathcal{E}\text{-AC}(\Sigma^+, \mathbb{E}^+)$ regarded as functors, we have $\overline{\mathcal{B}}(T\mathcal{A}(f)) = T(\mathcal{B} \odot \mathcal{A})(f)$.

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

4.1 Descriptive Composition for Nondeterministic MAGs

We look at an example of an extension of TSMCs arising from a symmetric monoidal monad $(T, \eta, \mu, \phi_1, \phi)$ over **Set**.³ 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 operation is defined by the following scheme:

$$id_A^{T\mathbb{C}} = 1 \xrightarrow{\phi_1} T1 \xrightarrow{T(id_A^{\mathbb{C}})} T_*(\mathbb{C})(A, A)$$

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

This construction is well-known as *change-of-base* in enriched category theory.

Proposition 3. *Let $T : \mathbf{Set} \rightarrow \mathbf{Set}$ be a monoidal monad. Then the mapping $\mathbb{C} \mapsto T_*\mathbb{C}$ extends to a monad T_* over \mathbf{TSMC}_s such that $T_* \circ \mathbf{Int} = \mathbf{Int} \circ T_*$.*

Proof. It is routine to check that $T_*(\mathbb{C})$ is a TSMC when \mathbb{C} is so. We show that $\mathbf{Int}(T_*(\mathbb{C})) = T_*(\mathbf{Int}(\mathbb{C}))$. First, the collection of objects in both categories are equal; just pairs of \mathbb{C} -objects. Next, both categories have identical homsets because

$$\begin{aligned} \mathbf{Int}(T_*(\mathbb{C}))((A^-, A^+), (B^-, B^+)) &= T_*(\mathbb{C})(A^+ \otimes B^-, B^+ \otimes A^-) \\ &= T(\mathbb{C}(A^+ \otimes B^-, B^+ \otimes A^-)) \\ &= T(\mathbf{Int}(\mathbb{C}))((A^-, A^+), (B^-, B^+)). \end{aligned}$$

The compact closed structures in $\mathbf{Int}(T_*(\mathbb{C}))$ and $T_*(\mathbf{Int}(\mathbb{C}))$ are the same.

From this, we obtain the Eilenberg-Moore adjunction $F \dashv U : \mathbf{TSMC}_s^{T_*} \rightarrow \mathbf{TSMC}_s$, and the **Int** construction lifts over $\mathbf{TSMC}_s^{T_*}$ thanks to $\mathbf{Int} \circ U = U \circ \mathbf{Int}$. Hence we obtain an extension $(F \dashv U, \mathbf{Int})$ of TSMCs.

Example 1. We write $\mathcal{P} : \mathbf{Set} \rightarrow \mathbf{Set}$ for the covariant powerset monad. Let $\mathbb{C} \in \mathbf{TSMC}_s$. A monoidal attribute grammar (I, S, a) for Σ in $\mathcal{P}_*(\mathbb{C})$ assigns \mathbb{C} -objects $I\rho, S\rho$ to each $\rho \in \Sigma$, 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 $o \in \Sigma^{\rho_1 \cdots \rho_n \rightarrow \rho}$; this means that a \mathcal{P}_* -MAG assigns an attribute calculation rule *nondeterministically* to each operator in Σ . For instance, we consider a \mathcal{P}_* -AC from Σ_{nat} to Σ_{list} determined by

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

$$a_Z = \{\emptyset\}, \quad a_S = \{(\lambda x . \text{let } y = 0 :: x \text{ in } y), (\lambda x . \text{let } y = 1 :: x \text{ in } y)\}$$

This AC maps each natural number $S^{(n)}(Z)$ to the set of all binary digit with length n , expressed as a morphism Σ_{list} of type $\epsilon \rightarrow *$.

³ This is equivalent to a commutative monad [14].

4.2 Descriptive Composition for quasi-SSUR ACs

In [18], a relaxation of the syntactic single use restriction called *quasi-SSUR* is given. In such ACs, the usage restriction of variables is changed from “exactly once” to “at most once”. We give a categorical account of the descriptive composition for quasi-SSUR ACs in the framework of extensions of TSMCs.

We extend the correspondence between SSUR-ACs and MAGs in free TSMCs to quasi-SSUR ACs. We introduce a new language $\mathcal{A}(\Sigma)$ by 1) modifying the typing rules of $\mathcal{L}(\Sigma)$ so that variables can be discarded, and 2) adding a constant \perp_ρ for each $\rho \in \Sigma$.

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

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

The point is that some defined variables may not be used ($U \subseteq D$). Because of this modification, the following is now a well-typed expression in the new type system:

$$\vdash_M \lambda \mathbf{x} . \text{let } \epsilon \text{ in } \epsilon : \rho \rightarrow \epsilon \quad (|\mathbf{x}| = |\rho|).$$

We write this expression \top_ρ .

2. We add to $\mathcal{L}(\Sigma)$ a constant $\perp_\rho : \rho$, denoting “undefined”, for each $\rho \in \Sigma$. We then extend this to any sequence of Σ -types by

$$\perp_{\rho_1, \dots, \rho_n} = (\lambda \epsilon . \text{let } x_1 = \perp_{\rho_1}, \dots, x_n = \perp_{\rho_n} \text{ in } x_1, \dots, x_n).$$

We define quasi-SSUR ACs as ACs over $\mathcal{A}(\Sigma)$ instead of $\mathcal{L}(\Sigma)$.

Definition 4. 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 $o \in \Delta^{\rho_1, \dots, \rho_n \rightarrow \rho}$, a_o is an $\mathcal{A}(\Sigma)$ -expression of type $a_o : S\rho_1, \dots, S\rho_n, I\rho \rightarrow S\rho, I\rho_1, \dots, I\rho_n$.

We next introduce *bipointed traced strict SMC*. It is a triple $(\mathbb{C}, \top, \perp)$ where $\mathbb{C} \in \mathbf{TSMC}_s$ and \top, \perp are \mathbb{C} -object indexed families of morphisms $\top_A : A \rightarrow \mathbf{I}$ and $\perp_A : \mathbf{I} \rightarrow A$ such that

$$\top_{\mathbf{I}} = \text{id}_{\mathbf{I}}, \quad \top_{A \otimes B} = \top_A \otimes \top_B, \quad \perp_{\mathbf{I}} = \text{id}_{\mathbf{I}}, \quad \perp_{A \otimes B} = \top_A \otimes \top_B. \quad (9)$$

We write \mathbf{TSMC}_s^\bullet for the category of bipointed traced strict SMCs and traced strict symmetric monoidal functors preserving bipoints. There is an evident forgetful functor $U^\bullet : \mathbf{TSMC}_s^\bullet \rightarrow \mathbf{TSMC}_s$, and it has a left adjoint $F^\bullet : \mathbf{TSMC}_s \rightarrow \mathbf{TSMC}_s^\bullet$ that freely adds morphisms $\perp_A : \mathbf{I} \rightarrow A$ and $\top_A : A \rightarrow \mathbf{I}$ for each object A , subject to the equations in (9). Furthermore, the **Int** construction can be *lifted* over \mathbf{TSMC}_s^\bullet . For a bipointed strict TSMC $(\mathbb{C}, \perp, \top)$, we define **Int**(\mathbb{C})-morphisms $\perp'_{(A^-, A^+)}$ and $\top'_{(A^-, A^+)}$ by

$$\perp'_{(A^-, A^+)} = \perp_{A^+} \otimes \top_{A^-}, \quad \top'_{(A^-, A^+)} = \top_{A^+} \otimes \perp_{A^-}.$$

Then we define **Int** $^\bullet(\mathbb{C}, \perp, \top)$ to be the tuple $(\mathbf{Int}(\mathbb{C}), \perp', \top')$. One can easily check that this is a bipointed traced strict SMC.

Proposition 4. *The mapping $(\mathbb{C}, \perp, \top) \mapsto \mathbf{Int}^*(\mathbb{C}, \perp, \top)$ extends to a lifting of \mathbf{Int} across U^* . Thus the triple $\mathcal{E}^* = (F^* \dashv U^*, \mathbf{Int}^*)$ forms an extension of TSMCs.*

Proposition 5. *The tuple $(\mathcal{A}(\Sigma), \{\perp_\rho\}_{\rho \in \Sigma^*}, \{\top_\rho\}_{\rho \in \Sigma^*})$ is a bipointed strict TSMC, and it is isomorphic to $F(\mathcal{L}(\Sigma))$.*

Corollary 1. *Quasi-SSURACs are \mathcal{E}^* -ACs.*

4.3 Descriptive Composition for Stack AGs

In [16], 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 descriptive composition. Inspired by his extension, below we give a corresponding categorical extension of ACs with stacks by setting-up an appropriate extension of TSMCs. Our approach differs from Nakano's work as follows: 1) the concept of stack ACs given below allow stacks to be stack elements, and stack elements to contain tuples rather than single element, and 2) we represent the empty stack by undefined stack \perp , and combine stack destructors (head, tail) into single operator \mathbf{dec} .

Below we introduce a language $\mathcal{S}(\Sigma)$ and capture the stack ACs satisfying the linearity condition as AGs in $\mathcal{S}(\Sigma)$. First, we define the set $|\mathcal{S}(\Sigma)|$ of types of $\mathcal{S}(\Sigma)$ to be the set of finite sequences of Σ -types and stack types C^∞ . Stack types can be nested, but C can not be the empty sequence. Its formal definition is:

$$|\mathcal{S}(\Sigma)| = |\mathcal{S}(\Sigma)|_0^* \quad |\mathcal{S}(\Sigma)|_0 \ni B ::= \rho \mid C^\infty \quad (\rho \in \Sigma, C \in |\mathcal{S}(\Sigma)|_0^+).$$

For $C \in |\mathcal{S}(\Sigma)|$, by $\mathbf{x} : C$ we mean the typing environment $x_1 : B_1, \dots, x_n : B_n$ where $B_i \in |\mathcal{S}(\Sigma)|_0$. The set of terms is defined by

$$M ::= \lambda \mathbf{x} . \text{let } D, \dots, D \text{ in } \mathbf{x}$$

$$D ::= x = x \mid x = o(x, \dots, x) \mid x = \perp_C \mid x = \mathbf{cons}_C(\mathbf{x}, x) \mid \mathbf{x}, x = \mathbf{dec}_C(x)$$

where o ranges over operators in Σ and C over $|\mathcal{S}(\Sigma)|_0^+$. The typing rules of $\mathcal{S}(\Sigma)$ extends $\mathcal{A}(\Sigma)$ with \mathbf{cons} and \mathbf{dec} :

$$\frac{B \in |\mathcal{S}(\Sigma)|_0}{x : B \vdash_E x : B} \quad \frac{B \in |\mathcal{S}(\Sigma)|_0}{\vdash_E \perp_B : B} \quad \frac{o \in \Sigma^{\rho_1 \dots \rho_n \rightarrow \rho}}{x_1 : \rho_1, \dots, x_n : \rho_n \vdash_E o(x_1, \dots, x_n) : \rho}$$

$$\frac{C \in |\mathcal{S}(\Sigma)| \quad C \neq \epsilon}{\mathbf{x} : C, y : C^\infty \vdash_E \mathbf{cons}_C(\mathbf{x}, y) : C^\infty} \quad \frac{C \in |\mathcal{S}(\Sigma)| \quad C \neq \epsilon}{x : C^\infty \vdash_E \mathbf{dec}_C(x) : C, C^\infty}$$

$$\frac{\Gamma_1 \vdash_E M_1 : C_1 \quad \dots \quad \Gamma_l \vdash_E M_l : C_l \quad U \subseteq D}{\vdash_M \lambda \mathbf{x} . \text{let } y_1 = M_1, \dots, y_l = M_l \text{ in } \mathbf{z} : C' \rightarrow C''}$$

where $U = \Gamma_1 \cup \dots \cup \Gamma_n \cup \{\mathbf{z} : C''\}$ and $D = \{\mathbf{x} : C', y_1 : C_1, \dots, y_l : C_l\}$, and each variable in $\Gamma_1, \dots, \Gamma_n, \mathbf{x}, \mathbf{y}, \mathbf{z}$ is different from the other. The set of axioms for $\mathcal{S}(\Sigma)$ -expressions extends the one for $\mathcal{A}(\Sigma)$ with

$$\text{let } \mathbf{x}, y = \mathbf{dec}_C(z), z = \mathbf{cons}_C(\mathbf{x}', y'), D \text{ in } \mathbf{v} = \text{let } \mathbf{x} = \mathbf{x}', y = y', D \text{ in } \mathbf{v}$$

$$\text{let } \mathbf{x}, y = \mathbf{dec}_C(z), z = \perp, D \text{ in } \mathbf{v} = \text{let } \mathbf{x} = \perp, y = \perp, D \text{ in } \mathbf{v}$$

$$\text{let } \mathbf{x}, y = \mathbf{cons}_C(z), D \text{ in } \mathbf{v} = \text{let } D \text{ in } \mathbf{v} \quad (\mathbf{x}, y \notin FV(D) \cup \mathbf{v}, z \neq y).$$

where C ranges over $|\mathcal{S}(\Sigma)|_0^+$.

Definition 5. 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 $o \in \Delta^{\rho_1, \dots, \rho_n \rightarrow \rho}$, a_o is an $\mathcal{S}(\Sigma)$ -expression of type $a_o : S\rho_1, \dots, S\rho_n, I\rho \rightarrow S\rho, I\rho_1, \dots, I\rho_n$.

Example 2. This example is from [16]. We consider a quasi-SSUR stack AC that converts reverse-polish notations to ordinary expressions. We consider two signatures

$$\begin{aligned}\Sigma_p &= \{ \{ * \}, \{ \text{push}_n^{* \rightarrow *}, \text{add}^{* \rightarrow *}, \text{mul}^{* \rightarrow *}, \text{ret}^{* \rightarrow *} \} \} \quad (n \in \mathbf{N}), \\ \Sigma_e &= \{ \{ * \}, \{ \text{num}_n^{* \rightarrow *}, \text{add}^{* \rightarrow *}, \text{mul}^{* \rightarrow *} \} \} \quad (n \in \mathbf{N}).\end{aligned}$$

The signature Σ_p is for the reverse-polish notation of expressions. For instance,

$$\text{push}_3(\text{push}_2(\text{push}_5(\text{add}(\text{mul}(\text{ret}))))))$$

denotes $(5+2)*3$. The stack AC (I, S, a) constructing Σ_e -terms corresponding to reverse-polish expressions is the following (type annotations are omitted):

$$\begin{aligned}I* &= (*)^\infty, O* = * \\ a_{\text{num}_n} &= \lambda s_1, i. \text{let } i_1 = \text{cons}(\text{num}_n, i) \text{ in } s_1, i_1 \\ a_{\text{add}} &= \lambda s_1, i. \text{let } h_1, t_1 = \text{dec}(i), h_2, t_2 = \text{dec}(t_1), i_1 = \text{cons}(\text{add}(h_1, h_2), t_2) \text{ in } s_1, i_1 \\ a_{\text{mul}} &= \lambda s_1, i. \text{let } h_1, t_1 = \text{dec}(i), h_2, t_2 = \text{dec}(t_1), i_1 = \text{cons}(\text{mul}(h_1, h_2), t_2) \text{ in } s_1, i_1 \\ a_{\text{ret}} &= \lambda i. \text{let } \epsilon \text{ in } i.\end{aligned}$$

Definition 6. A traced strict SMC with stack is a tuple $(\mathbb{C}, \perp, \top, (-)^\infty, \text{cons}, \text{dec})$ where

- $(\mathbb{C}, \perp, \top)$ is a bipointed SMC,
- $(-)^\infty : |\mathbb{C}| \rightarrow |\mathbb{C}|$ is a mapping and
- $\text{cons}_X : X \otimes X^\infty \rightarrow X^\infty$ and $\text{dec}_X : X^\infty \rightarrow X \otimes X^\infty$ are \mathbb{C} -object indexed families of morphisms such that

$$\begin{aligned}\text{dec}_X \circ \text{cons}_X &= \text{id}_{X \otimes X^\infty}, & \text{dec}_{\mathbf{I}} &= \text{cons}_{\mathbf{I}} = \text{id}_{\mathbf{I}}, \\ \text{dec}_X \circ \perp_X &= \perp_{X^\infty} \otimes \perp_{X^\infty}, & \top_X \circ \text{cons}_X &= \top_{X^\infty} \otimes \top_{X^\infty}.\end{aligned}$$

We write \mathbf{TSMC}_s^S for the category of small traced strict SMCs with stack and traced strict symmetric monoidal functors preserving $\perp, \top, \text{cons}, \text{dec}$.

There is a canonical forgetful functor $U^S : \mathbf{TSMC}_s^S \rightarrow \mathbf{TSMC}_s$, and this has a left adjoint $F^S : \mathbf{TSMC}_s \rightarrow \mathbf{TSMC}_s^S$. Next, let $\mathbb{C}^S = (\mathbb{C}, \perp, \top, (-)^\infty, \text{cons}, \text{dec}) \in \mathbf{TSMC}_s^S$. We define a mapping $(-)^\infty'$ over $\mathbf{Int}(\mathbb{C})$ -objects and $\mathbf{Int}(\mathbb{C})$ -morphisms as follows:

$$\begin{aligned}(A^-, A^+)^\infty' &= ((A^-)^\infty, (A^+)^\infty) \\ \perp'_{(A^-, A^+)} &= \perp_{A^+} \otimes \top_{A^-}, & \top'_{(A^-, A^+)} &= \top_{A^+} \otimes \perp_{A^-}, \\ \text{cons}'_{(A^-, A^+)} &= \text{cons}_{A^+} \otimes \text{dec}_{A^-}, & \text{dec}'_{(A^-, A^+)} &= \text{dec}_{A^+} \otimes \text{cons}_{A^-}.\end{aligned}$$

We then write $\mathbf{Int}^S(\mathbb{C}^S)$ for the tuple $(\mathbf{Int}(\mathbb{C}), \perp', \top', (-)^\infty', \text{cons}', \text{dec}')$.

Proposition 6. The mapping $\mathbb{C}^S \mapsto \mathbf{Int}^S(\mathbb{C}^S)$ extends to a lifting of \mathbf{Int} across U^S . Thus the triple $\mathcal{E}^S = (F^S \dashv U^S, \mathbf{Int}^S)$ forms an extension of TSMCs.

Proposition 7. *The tuple $(\mathcal{S}(\Sigma), \perp, \top, (-)^\infty, \text{cons}, \text{dec})$ is a traced strict SMC with stack, and is isomorphic to $F^S(\mathcal{L}(\Sigma))$.*

Corollary 2. *Quasi-SSUR stack ACs are \mathcal{E}^S -ACs, and they are closed under the descriptonal composition.*

5 Conclusion

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

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

References

1. Miklós Bartha. Linear deterministic attributed transformations. *Acta Cybern.*, 6:125–147, 1983.
2. John Boyland. Conditional attribute grammars. *ACM Trans. Program. Lang. Syst.*, 18(1):73–108, 1996.
3. Laurian M. Chirica and David F. Martin. An order-algebraic definition of knuthian semantics. *Mathematical Systems Theory*, 13:1–27, 1979.
4. 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.
5. Harald Ganzinger. Increasing modularity and language-independency in automatically generated compilers. *Sci. Comput. Program.*, 3(3):223–278, 1983.
6. Harald Ganzinger and Robert Giegerich. Attribute coupled grammars. In *SIGPLAN Symposium on Compiler Construction '84*, pages 157–170. ACM, 1984.
7. Robert Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Inf.*, 25(4):355–423, 1988.
8. Masahito Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of let and letrec*. Springer-Verlag, 1999.
9. 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.
10. Andre Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.
11. Shinya Katsumata. Attribute grammars and categorical semantics. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 271–282. Springer, 2008.
12. Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

13. Donald E. Knuth. Correction: Semantics of context-free languages. *Mathematical Systems Theory*, 5(1):95–96, 1971.
14. Anders Kock. Strong functors and monoidal monads. *Archiv der Math.*, 23(1):113–120, 1972.
15. Saunders MacLane. *Categories for the Working Mathematician (Second Edition)*, volume 5 of *Graduate Texts in Mathematics*. Springer, 1998.
16. Keisuke Nakano. Composing stack-attributed tree transducers. *Theory Comput. Syst.*, 44(1):1–38, 2009.
17. Susumu Nishimura. Deforesting in accumulating parameters via type-directed transformations. In *APLAS '02*, pages 145–159, 2002.
18. Susumu Nishimura and Keisuke Nakano. XML stream transformer generation through program composition and dependency analysis. *Sci. Comput. Program.*, 54(2-3):257–290, 2005.
19. Janis Voigtländer. Using circular programs to deforest in accumulating parameters. *Higher-Order and Symbolic Computation*, 17(1-2):129–163, 2004.

A Traced Symmetric Monoidal Categories and the **Int** Construction

We recall the concept of traced symmetric monoidal categories and **Int** construction by Joyal et al [10]. Below we mainly consider *strict* symmetric monoidal categories for legibility. A *trace operator* on a symmetric monoidal category $(\mathbb{C}, \mathbf{I}, \otimes, \sigma)$ is a mapping $\text{tr}_{B,C}^A : \mathbb{C}(B \otimes A, C \otimes A) \rightarrow \mathbb{C}(B, C)$ satisfying the following equations: We simplify

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

Fig. 6. Axioms for Trace Operators

the superposing axiom in [10] using naturality and dinaturality [8]. A *traced symmetric monoidal category (TSMC)* is a pair of a symmetric monoidal category (SMC) and a trace operator on it. We say that a strict symmetric monoidal functor $F : \mathbb{C} \rightarrow \mathbb{D}$ between TSMCs \mathbb{C}, \mathbb{D} is *traced* if F preserves traces, that is, $F \text{tr}(f) = \text{tr}(Ff)$.

Joyal et al. gave a free construction, called **Int**, of tortile monoidal categories from traced monoidal categories. In this paper, we regard this construction as a monad on \mathbf{TSMC}_s . Let \mathbb{C} be a TSMC. We define the category $\mathbf{Int}(\mathbb{C})$ by the following data. An object is a pair (A^+, A^-) of \mathbb{C} -objects, and a morphism from (A^+, A^-) to (B^+, B^-) is a \mathbb{C} -morphism $f : A^+ \otimes B^- \rightarrow B^+ \otimes A^-$. The composition of $\mathbf{Int}(\mathbb{C})$ -morphisms $f : (A^+, A^-) \rightarrow (B^+, B^-)$ and $g : (B^+, B^-) \rightarrow (C^+, C^-)$ is define by the following trace:

$$g \circ f = \text{tr}_{A^+ \otimes C^-, C^+ \otimes A^-}^{B^-}((\text{id} \otimes \sigma) \circ (g \otimes \text{id}) \circ (\text{id} \otimes \sigma) \circ (f \otimes \text{id}) \circ (\text{id} \otimes \sigma)).$$

Category $\mathbf{Int}(\mathbb{C})$ has the following symmetric monoidal structure:

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

The trace of $f : A \otimes C \rightarrow B \otimes C$ is given as follows:

$$\text{tr}(f) = \text{tr}_{A^+ \otimes B^-, B^+ \otimes A^-}^{C^+ \otimes C^-}((\text{id}_{B^+} \otimes \sigma_{C^+, A^-} \otimes \text{id}_{C^-}) \circ f \circ (\text{id}_{A^+} \otimes \sigma_{B^-, C^+} \otimes \text{id}_{C^-}))$$

The following theorem is a consequence of the structure theorem for traced symmetric monoidal categories [10].

Theorem 3. *The mapping $\mathbb{C} \mapsto \mathbf{Int}(\mathbb{C})$ extends to a monad on \mathbf{TSMC}_s .*