A New Foundation of Attribute Grammars in Traced Symmetric Monoidal Categories

Shin-ya Katsumata
Research Institute for Mathematical Sciences, Kyoto University

July 23, 2007

Abstract

In this paper we propose a new categorical formulation of *attribute grammars* in *traced symmetric monoidal categories*. The new formulation, called *monoidal attribute grammars*, concisely captures the essence of the classical attribute grammars. We study monoidal attribute grammars in two categories: $\mathbf{Rel}^+$ and $\omega\mathbf{CPPO}$. It turns out that in $\mathbf{Rel}^+$ monoidal attribute grammars correspond to the graph-theoretic representation of attribute dependencies, while in $\omega\mathbf{CPPO}$ monoidal attribute grammars are equivalent to Chirica and Martin's K-systems. We also show that in traced symmetric monoidal closed categories every monoidal attribute grammar is equivalent to the one which does not use inherited attributes.

1 Introduction

Attribute grammars [25, 26] are a mechanism to assign bidirectional computation to derivation trees of context free grammars (CFG for short). There are two types of in-

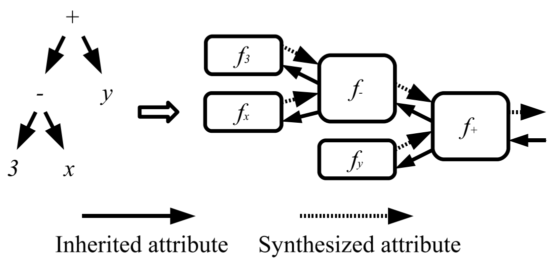


Figure 1: Attribute Grammar

formation flowing along derivation trees: *inherited attributes* that are propagated from the top to the bottom of trees, and *synthesized attributes* that are built-up toward the top from the bottom of trees. This feature brought great flexibility to attribute grammars, and lead them to a big success in applications, such as compiler constructions [13, 14, 11, 5, 29] and program transformations [7, 8, 30, 28].

The main purpose of this paper is to give a categorical formulation of attribute grammars. In general, attribute grammars may assign *cyclic computation* of attributes to derivation trees, like Figure 2, and handling such situation is an unavoidable issue in

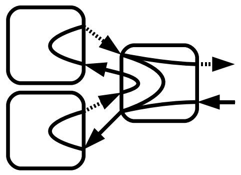


Figure 2: Cyclic Computation of Attributes

formulating attribute grammars. We tackle this issue by employing traced symmetric monoidal categories [22] as underlying semantic categories so that cyclic computation of attributes is naturally absorbed by trace operators.

The actual formulation of attribute grammars is carried out in the categories obtained by Joyal, Street and Verity's **Int** construction [22] (equivalently Abramsky's $\mathcal{G}$ construction [1]). The basic idea of this formulation is that the category obtained by **Int** construction from a traced symmetric monoidal category provides an ideal platform for modeling bidirectional computation, which we exactly need to model attribute grammars.

The new formulation of attribute grammars is called *monoidal attribute grammars*. They are a monoidal version of the classical attribute grammars, and have the following advantages: i) we no longer need to stick to set-theoretic representation of attribute domains and semantic rules, and ii) any attribute grammars, including those which assign cyclic computation of attributes, are modeled at a highly abstract level.

After monoidal attribute grammars are introduced in Section 5, we devote the rest of the paper to the verification of the formulation.

- In Section 6 we point out a connection between local dependency graphs of the classical attribute grammars and monoidal attribute grammars in $\mathbf{Rel}^+$. This connection shed light on Knuth's algorithm [26] for detecting well-formedness from a different angle.

- In Section 7 we show that Chirica and Martin's *K-systems* [6], which are a domain-theoretic formulation of attribute grammars, are equivalent to monoidal attribute grammars in $\omega\mathbf{CPPO}$. This is done by establishing semantics-preserving translations between these two formulations.

- In Section 8 we show that every monoidal attribute grammar in a traced symmetric monoidal closed category is equivalent to the one which does not use inherited attributes. This equivalence is a generalization of Chirica and Martin's observation on attribute grammars as initial algebra homomorphisms [6].

2 Preliminaries

A *many-sorted signature* Σ is a pair (S, O) of the set S of *sorts* and S^+-indexed family O of sets of *operators*. For a signature $\Sigma = (S, O)$, by $s \in \Sigma$ and $o \in \Sigma^{s_1 \cdots s_n \to s}$ we mean $s \in S$ and $o \in O^{s_1 \cdots s_n s}$, respectively.

In this paper we represent derivation trees of a CFG as closed terms of the many sorted signature associated to the CFG. Let $G = (T, N, P, S)$ be a CFG. We associate to it a many-sorted signature $\Sigma_G = (N, \overline{P})$ where $\overline{P}^{X_1 \cdots X_n X}$ is the collection of production rules of the form

$$X \to a_0 X_1 a_1 \cdots X_n a_n$$

with some terminal strings $a_0, \cdots, a_n \in T^*$.

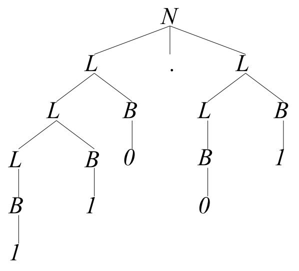
We use the following CFG throughout this paper:

$$\begin{aligned} G_B \quad = \quad & (\{0,1\}, \{B, L, N\}, \\ & \{p_1 : B \to 0, p_2 : B \to 1, \\ & \ p_3 : L \to B, p_4 : L \to LB, \\ & \ p_5 : N \to L, p_6 : N \to L.L\}, L). \end{aligned}$$

The language of this CFG is binary representations of rational numbers, like 110.01. The many-sorted signature Σ_{G_B} associated to G_B is:

$$\Sigma_{G_B} = (\{B, L, N\}, \{p_1^B, p_2^B, p_3^{B \to L}, p_4^{L,B \to L}, p_5^{L \to N}, p_6^{L,L \to N}\}).$$

The derivation trees of G_B whose root is a nonterminal symbol $X \in N$ will be represented by closed Σ_{G_B}-terms of sort X. For instance, the following derivation tree of G_B:



is represented by the following closed Σ_{G_B}-term of $N \in \Sigma_{G_B}$:

$$p_6(p_4(p_4(p_3(p_2), p_2), p_1), p_4(p_3(p_1), p_2)).$$

Classical Attribute Grammars

In the seminal paper [25] published in 1968, Knuth proposed attribute grammars as a framework for the semantics of CFGs. Attribute grammars assign computation of values called *attributes* to derivation trees. There are two types of attributes that flow along derivation trees in different directions; *inherited attributes* that are propagated from the top to the bottom of derivation trees, and *synthesized attributes* that are built-up from the bottom to the top of trees. We recall the classical definition of attribute grammars [25, 10].

Definition 2.1 A *(classical) attribute grammar* for a CFG $G = (T, N, P, S)$ is a tuple $(\mathbf{Inh}, \mathbf{Syn}, V, f)$ such that

- $\mathbf{Inh}, \mathbf{Syn}$ are N-indexed families of finite sets of *inherited* and *synthesized attribute names*, respectively. We assume that $\mathbf{Inh}\,X$ and $\mathbf{Syn}\,X$ are disjoint.
- V is a family $\{V_{X.a}\}$ of sets called *attribute domain*, where $X \in N$ and $a \in \mathbf{Inh}\,X \cup \mathbf{Syn}\,X$. For each nonterminal symbol $X \in N$, we define

$$V_{\mathbf{Inh}\,X} = \prod_{a \in \mathbf{Inh}\,X} V_{X.a}, \qquad V_{\mathbf{Syn}\,X} = \prod_{a \in \mathbf{Syn}\,X} V_{X.a},$$

$$V_X = V_{\mathbf{Inh}\,X} \times V_{\mathbf{Syn}\,X}.$$

- f is a P-indexed family of functions called *semantic rules*, such that for each production rule $p : X \to a_0 X_1 a_1 \cdots X_{n_p} a_{n_p} \in P$ the type of f_p is

$$f_p : \left(\prod_{k=1}^{n_p} V_{\mathbf{Syn}\,X_k} \right) \times V_{\mathbf{Inh}\,X} \to V_{\mathbf{Syn}\,X} \times \prod_{k=1}^{n_p} V_{\mathbf{Inh}\,X_k}. \tag{1}$$

  It represents the relationship between incoming and outgoing attributes at the node corresponding to production rule p.

Following [25, 10], we assume that terminal symbols do not contribute to computation of attributes. On the other hand, in this paper we do not explicitly assume $\mathbf{Inh}\,S = \emptyset$, which is a typical assumption in the classical attribute grammars, as it does not have any substantial effect to the categorical formulation of attribute grammars.

We aim to categorically formulate attribute grammars of Definition 2.1. In some literatures semantic rules are relaxed to functions of the following type:

$$f_p : V_X \times \prod_{k=1}^{n_p} V_{X_k} \to V_{\mathbf{Syn}\,X} \times \prod_{k=1}^{n_p} V_{\mathbf{Inh}(X_k)}. \tag{2}$$

We call such attribute grammars *full*. We do not study full attribute grammars directly; instead, in Section 7 we discuss that full attribute grammars can be reduced to normal ones by applying fixpoint operators to semantic rules.

The following is an example of the classical attribute grammar U_B for G_B in [25].

- Attribute names:

$$\begin{aligned} \mathbf{Inh}\,B &= \{s\} & \mathbf{Syn}\,B &= \{v\} \\ \mathbf{Inh}\,L &= \{s\} & \mathbf{Syn}\,L &= \{v, l\} \\ \mathbf{Inh}\,N &= \emptyset & \mathbf{Syn}\,N &= \{v\} \end{aligned}$$

- Attribute domains:

$$V_{B.s} = \mathbf{Z}, V_{B.v} = \mathbf{Q}$$

$$V_{L.v} = \mathbf{Q}, V_{L.l} = \mathbf{Z}, V_{L.s} = \mathbf{Z}$$

$$V_{N.v} = \mathbf{Q}$$

- Semantic rules:

$$\begin{array}{rcl} f_{p_0}(n) & = & 0 \\ f_{p_1}(n) & = & 2^n \\ f_{p_2}(x,y) & = & (y,1,x) \\ f_{p_3}(x,y,z,w) & = & (y+w, z+1, x+1, x) \\ f_{p_4}(x,y) & = & (x,0) \\ f_{p_5}(x,y,z,w) & = & (x+z, 0, -w) \end{array}$$

This attribute grammar converts sentences of L_B (which are binary representations of rational numbers) to rational numbers.

3 Categorical Semantics of Parse Trees

We first consider assigning one-way computation of values to derivation trees of CFGs. Such computation may be modeled by various mathematical structures, such as functions over sets, continuous functions over domains, relations over sets, etc. In this paper, instead of committing into a particular mathematical structure, we employ *monoidal categories* [27] for an abstract and categorical representation of computation.

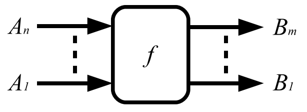
A monoidal category consists of a category $\mathbb{C}$, an object $\mathbf{I}$ in $\mathbb{C}$ called *unit*, a bifunctor $\otimes : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ called *tensor* and natural isomorphisms

$$l_A : \mathbf{I} \otimes A \to A, \qquad r_A : A \otimes \mathbf{I} \to A$$

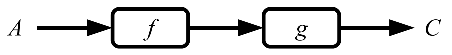
$$a_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$$

subject to certain coherence axioms; see [27] for detail. A monoidal category is called *strict* if the natural isomorphisms l, r, a are all identity morphisms. So in strict monoidal categories we have $(A \otimes B) \otimes C = A \otimes (B \otimes C), \mathbf{I} \otimes A = A \otimes \mathbf{I} = A$. Every monoidal category is equivalent to a strict one (coherence theorem [27]), so we mainly talk about strict monoidal categories for legibility.

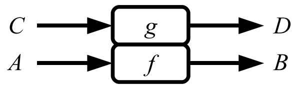
We introduce a geometric representation of morphisms in $\mathbb{C}$. The principle of the representation is that we depict a morphism $f : A_1 \otimes \cdots \otimes A_n \to B_1 \otimes \cdots \otimes B_m$ as a circuit which has n-inputs on the left and m-outputs on the right:



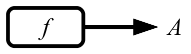
We depict the composition $g \circ f$ of $f : A \to B$ and $g : B \to C$ by connecting f's output to g's input:



while the tensor $f \otimes g$ of $f : A \to B$ and $g : C \to D$ by stacking f on the top of g:



A morphism $f : \mathbf{I} \to A$ from the unit is depicted as a circuit with no input:



Such morphisms can be regarded as an element or a constant in A, so we call a morphism $f : \mathbf{I} \to A$ *global element* of A. A mathematical justification of the above geometric representation is given in [21].

Definition 3.1 Let $\Sigma = (S, O)$ be a signature and $\mathbb{C}$ be a monoidal category.

1. A Σ-*algebra* in $\mathbb{C}$ is a pair (A, α) such that A is an S-indexed family of $\mathbb{C}$-objects and α is an operator-indexed family of $\mathbb{C}$-morphisms such that for each operator $o \in \Sigma^{s_1 \ldots s_n \to s}$, the type of α_o is
$$\alpha_o : As_1 \otimes \ldots \otimes As_n \to As.$$
2. Let $\mathcal{A} = (A, \alpha)$ and $\mathcal{B} = (B, \beta)$ be Σ-algebras in $\mathbb{C}$. A Σ-*homomorphism* from $\mathcal{A}$ to $\mathcal{B}$ is a S-indexed family h of $\mathbb{C}$-morphisms such that for each sort $s \in \Sigma$, h_s is a morphism from As to Bs, and h satisfies
$$\beta_o \circ (h_{s_1} \otimes \ldots \otimes h_{s_n}) = h_s \circ \alpha$$
for each $o \in \Sigma^{s_1 \ldots s_n \to s}$.
3. We write $\mathbf{Alg}_\Sigma(\mathbb{C})$ for the category of Σ-algebras and Σ-algebra homomorphisms in $\mathbb{C}$.

We note that set-theoretic Σ-algebras are precisely captured by $\mathbf{Alg}_\Sigma(\mathbf{Set})$. We write $\mathcal{T}_\Sigma = (T_\Sigma, \iota)$ for the initial object in $\mathbf{Alg}_\Sigma(\mathbf{Set})$.

Definition 3.2 Let $\mathbb{C}$ be a monoidal category $\mathbb{C}$ and $\mathcal{A} = (A, \alpha)$ be a Σ-algebra in $\mathbb{C}$. The *interpretation* of a Σ-term t of sort $s \in \Sigma$ by $\mathcal{A}$ is the global element $\mathcal{A}[\![t]\!] : \mathbf{I} \to As$ defined by induction on the structure of t:

$$\frac{\mathcal{A}[\![t_1]\!] : \mathbf{I} \to As_1 \quad \cdots \quad \mathcal{A}[\![t_n]\!] : \mathbf{I} \to As_n \quad o \in \Sigma^{s_1 \cdots s_n \to s}}{\mathcal{A}[\![o(t_1, \cdots, t_n)]\!] = \alpha_p \circ (\mathcal{A}[\![t_1]\!] \otimes \cdots \otimes \mathcal{A}[\![t_n]\!]) : \mathbf{I} \to As}$$

At geometric presentation level, this interpretation assigns a one-way computation of values to a derivation tree of a CFG. For instance, the interpretation of a Σ_B-term $t = p_4(p_4(p_3(p_2), p_2), p_1)$ of sort L by a Σ_B-algebra (A, α) is the $\mathbb{C}$-morphism depicted as follows:

$$\begin{tikzcd}
\boxed{\alpha_{p_2}} \arrow[r] & \boxed{\alpha_{p_3}} \arrow[dr] & & & \\
& & \boxed{\alpha_{p_4}} \arrow[dr] & & \\
& \boxed{\alpha_{p_2}} \arrow[ur] & & \boxed{\alpha_{p_4}} \arrow[r] & AL \\
& & \boxed{\alpha_{p_1}} \arrow[ur] & &
\end{tikzcd}$$

An abstract aspect of $\mathbf{Alg}_\Sigma$ is that it determines a 2-functor. We first introduce the 2-category of monoidal categories, monoidal functors and monoidal natural transformations.

Definition 3.3 1. Let $\mathbb{C}, \mathbb{D}$ be monoidal categories. A *monoidal functor* from $\mathbb{C}$ to $\mathbb{D}$ consists of a functor $F : \mathbb{C} \to \mathbb{D}$, a morphism $m : \mathbf{I}^{\mathbb{D}} \to F\mathbf{I}^{\mathbb{C}}$ in $\mathbb{D}$ and a natural transformation $n_{A,B} : FA \otimes^{\mathbb{D}} FB \to F(A \otimes^{\mathbb{C}} B)$ subject to certain coherence axioms; see [27] for detail. A monoidal functor (F, m, n) is called *strong* if m, n are (natural) isomorphisms. We often refer to a monoidal functor (F, m, n) by the name of its functor part.

2. Let $(F, m^F, n^F), (G, m^G, n^G)$ be monoidal functors from $\mathbb{C}$ to $\mathbb{D}$. A *monoidal natural transformation* from F to G is a natural transformation ϕ from F to G such that ϕ respects the monoidal structure; see [27] for detail.

3. Small monoidal categories, monoidal functors and monoidal natural transformations form a 2-category, which we call by **Mon**.

Proposition 3.4 *The mapping* $\mathbb{C} \mapsto \mathbf{Alg}_\Sigma(\mathbb{C})$ *extends to a 2-functor from* **Mon** *to* **Cat***.*

Proof For a monoidal functor $(F, m, n) : \mathbb{C} \to \mathbb{D}$, we define a functor $\mathbf{Alg}_\Sigma(F) : \mathbf{Alg}_\Sigma(\mathbb{C}) \to \mathbf{Alg}_\Sigma(\mathbb{D})$ by

$$\mathbf{Alg}_\Sigma(F)(A, \alpha) = (FA, \overline{\alpha})$$

where $\overline{\alpha}$ is the operator-indexed family of $\mathbb{D}$-morphisms such that for each $o \in \Sigma^{1\ldots s_l \to s}$, $\overline{\alpha}_o$ is the following composition:

$$\overline{\alpha}_o : FAs_1 \otimes^{\mathbb{D}} \ldots \otimes^{\mathbb{D}} FAs_l \xrightarrow{n^l} F(As_1 \otimes^{\mathbb{C}} \ldots \otimes^{\mathbb{C}} As_l) \xrightarrow{F\alpha_o} FAs$$

where n^l is an extension of n to the tensor of multiple objects.

For a monoidal natural transformation $\alpha : (F, m^F, n^F) \to (G, m^G, n^G)$, we define a natural transformation $\mathbf{Alg}_\Sigma(\alpha) : \mathbf{Alg}_\Sigma(F) \to \mathbf{Alg}_\Sigma(G)$ by

$$(\mathbf{Alg}_\Sigma(\alpha)_{(A,\alpha)})_s = \alpha_{As}.$$

It is routine to check that the above data determines a 2-functor from **Mon** to **Cat**.

We can capture the inductive definition of the interpretation (Definition 3.2) in terms of *initial algebra semantics* in $\mathbf{Alg}_\Sigma(\mathbf{Set})$. For this, we first introduce the global element functor.

Definition 3.5 1. Let $\mathbb{C}$ be a monoidal category. We define the global element monoidal functor $G_{\mathbb{C}} : \mathbb{C} \to \mathbf{Set}$ by

$$G_{\mathbb{C}} = \mathbb{C}(\mathbf{I}, -) : \mathbb{C} \to \mathbf{Set}.$$

2. Let $\mathbb{C}, \mathbb{D}$ be monoidal categories and $(F, m^F, n^F) : \mathbb{C} \to \mathbb{D}$ be a monoidal functor. We define a monoidal natural transformation $G_F : G_{\mathbb{C}} \to G_{\mathbb{D}} \circ F$ by

$$(G_F)_C(f) = Ff \circ m^F.$$

Proposition 3.6 *The interpretation $\mathcal{A}[\![-]\!]$ coincides with the unique Σ-algebra homomorphism*

$$\mathcal{A}[\![-]\!] : \mathcal{T}_\Sigma \longrightarrow \mathbf{Alg}_\Sigma(G_{\mathbb{C}})(\mathcal{A}) \qquad \textit{in } \mathbf{Alg}_\Sigma(\mathbf{Set}).$$

Different Σ-algebras in different categories may give rise to isomorphic Σ-algebras in $\mathbf{Alg}_\Sigma(\mathbf{Set})$ via G. In this case we call these algebras *equivalent*.

Definition 3.7 We say that two Σ-algebras $\mathcal{A}$ in $\mathbb{C}$ and $\mathcal{B}$ in $\mathbb{D}$ are *equivalent* if $\mathbf{Alg}_\Sigma(G_{\mathbb{C}})(\mathcal{A})$ and $\mathbf{Alg}_\Sigma(G_{\mathbb{D}})(\mathcal{B})$ are isomorphic in $\mathbf{Alg}_\Sigma(\mathbf{Set})$.

4 Traced Symmetric Monoidal Categories and Int Construction

Attribute grammars assign computation of attributes that flow along derivation trees in two directions (Figure 1). At first sight the algebraic framework in the previous section seems not adequate for modeling attribute grammars. However, if we employ monoidal categories in which each morphism *intrinsically* model bidirectional computation, like Figure 3, then by instantiating the algebraic framework with such categories we obtain

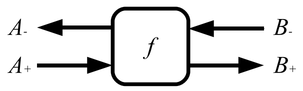


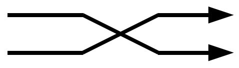
Figure 3:

categorical models of attribute grammars.

The categories that match with our purpose can be obtained by Joyal, Street and Verity's **Int** *construction* [22], or equivalently Abramsky's $\mathcal{G}$ construction [1] on *traced symmetric monoidal categories*. In this section we review traced symmetric monoidal categories and **Int** construction; for detailed exposition see [22, 1, 20].

4.1 Traced Symmetric Monoidal Categories

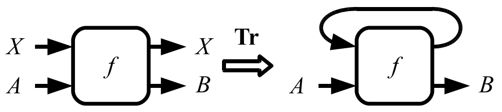
A *symmetry* on a monoidal category $\mathbb{C}$ is a natural isomorphism $c_{A,B} : A \otimes B \to B \otimes A$ subject to certain coherence axioms; see [27] for detail. A pair of a monoidal category and a symmetry on it is called *symmetric monoidal category* (*SMC* for short). We note that every (co-)Cartesian category is a SMC by fixing a (initial) terminal object and binary (co-)products. The geometric representation of symmetry is the *crossing* of wires.



Let $\mathbb{C}, \mathbb{D}$ be SMCs. A (strong) monoidal functor $(F, m, n) : \mathbb{C} \to \mathbb{D}$ is symmetric if F respects the symmetry in $\mathbb{C}$ in the obvious way (see [27] for detail).

Proposition 4.1 *For any SMC $\mathbb{C}$, the global element functor $G_{\mathbb{C}} : \mathbb{C} \to$* **Set** *is symmetric.*

Trace operators are introduced and studied in the seminal paper [22] by Joyal, Street and Verity [1]. Intuitively a trace operator **Tr** forms the feed-back loop at terminals X of $f : A \otimes X \to B \otimes X$:



Definition 4.2 ([22]) A *trace operator* on a SMC $\mathbb{C}$ is a family of mappings $\mathbf{Tr}^X_{A,B} : \mathbb{C}(A \otimes X, B \otimes X) \to \mathbb{C}(A, B)$ that satisfies the following axioms:

Vanishing(Unit) For any $f : A \otimes \mathbf{I} \to B \otimes \mathbf{I}$,

$$\mathbf{Tr}^{\mathbf{I}}_{A,B}(f) = f.$$

Vanishing(Tensor) For any $f : A \otimes (X \otimes Y) \to B \otimes (X \otimes Y)$,

$$\mathbf{Tr}^{X \otimes Y}_{A,B}(f) = \mathbf{Tr}^X_{A,B}(\mathbf{Tr}^Y_{A \otimes X, B \otimes X}(f)).$$

Superposing For any $f : A \otimes X \to B \otimes X$,

$$\mathbf{Tr}^X_{C \otimes A, C \otimes B}(C \otimes f) = C \otimes \mathbf{Tr}^X_{A,B}(f).$$

Yanking

$$\mathbf{Tr}^X_{X,X}(c_{X,X}) = \mathrm{id}_X.$$

Left Tightening For any $f : A' \otimes X \to B \otimes X$ and $g : A \to A'$,

$$\mathbf{Tr}^X_{A,B}(f \circ (g \otimes X)) = \mathbf{Tr}^X_{A',B}(f) \circ g.$$

Right Tightening For any $f : A \otimes X \to B \otimes X$ and $g : B \to B'$,

$$\mathbf{Tr}^X_{A,B'}((g \otimes X) \circ f) = g \circ \mathbf{Tr}^X_{A,B}(f).$$

Sliding For any $f : A \otimes X \to B \otimes X$ and $g : Y \to X$,

$$\mathbf{Tr}^X_{A,B}((B \otimes g) \circ f) = \mathbf{Tr}^Y_{A,B}(f \circ (B \otimes g)).$$

A *traced symmetric monoidal category* (*TSMC* for short) is a pair of a SMC with a trace operator on it. Similarly a *traced Cartesian category* (*TCC* for short) is a pair of a Cartesian category with a trace operator with respect to Cartesian products.

[1] Actually, trace operators are introduced to *balanced monoidal categories*, which are a generalization of SMCs.

The geometric representation of these axioms can be found in [22, 20, 2].

Definition 4.3 ([22]) Let $\mathbb{C}, \mathbb{D}$ be TSMCs. A *traced symmetric monoidal functor* is a symmetric strong monoidal functor $(F, m, n) : \mathbb{C} \to \mathbb{D}$ that preserves the trace operator in the following sense:

$$\mathbf{Tr}^{FX}_{FA,FB}(m^{-1}_{A,B} \circ Ff \circ m_{A,B}) = F(\mathbf{Tr}^{X}_{A,B}(f)) \qquad (f : A \otimes X \to B \otimes X)$$

Below we present some examples of TSMCs [22, 2].

Example 4.4 ([22]) The category **Rel** of sets and binary relations is Cartesian; a terminal object is given by the empty set, and binary products are given by disjoint unions. Hence **Rel** is a SMC.

We define a trace operator on this symmetric monoidal structure. Let $f : A + X \nrightarrow B + X$ be a binary relation. We decompose f into four relations

$$f_{AB} : A \nrightarrow B,\ f_{AX} : A \nrightarrow X,\ f_{XB} : X \nrightarrow B,\ f_{XX} : X \nrightarrow X$$

where f_{PQ} $(P = A/X, Q = B/X)$ is the subrelation of f such that f_{PQ} relates elements injected from P and Q. We then define a relation $\mathbf{Tr}(f)$ by

$$\mathbf{Tr}(f) = f_{AB} \cup (f_{XB} \circ f^{*}_{XX} \circ f_{AX})$$

(here f^{*}_{XX} denotes the transitive reflexive closure of f_{XX}). This relation satisfies the axioms of trace operators in Definition 4.2. Hence the above data form a TCC, which we write by $\mathbf{Rel}^{+}$.

Example 4.5 ([1]) We write ω**CPPO** for the category of ω-complete pointed partial orders and continuous functions between them. This category is Cartesian, hence is a SMC.

Let $f : [A \times X \to B \times X]$ be a continuous function. The following continuous function $\mathbf{Tr}(f) \in [A \to B]$:

$$\mathbf{Tr}(f)(a) = \pi_1(f(a, \mathbf{fix}(\lambda x . \pi_2(f(a, x))))),$$

where **fix** is the least fixpoint operator, satisfies the axioms of trace operators in Definition 4.2. Hence the above data form a TCC, which we write by ω**CPPO**.

The above construction is a part of the bijective correspondence between parameterized fixpoint operators and trace operators in TCCs. This is independently observed by Hasegawa and Hyland.

Theorem 4.6 ([20]) *In a Cartesian category $\mathbb{C}$, there is a bijective correspondence between trace operators and parameterized fixpoint operators $\mathbf{fix}^{X}_{A} : \mathbb{C}(A \times X, X) \to \mathbb{C}(A, X)$; see [20] for the axioms of* **fix***.*

4.2 Int construction

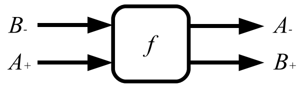
Joyal, Street and Verity's **Int** construction [22] yields a category in which every morphism represents bidirectional computation. An isomorphic construction, called $\mathcal{G}$ *construction*, is also considered by Abramsky for the analysis of Girard's geometry-of-interaction [1, 3]. Their geometric representation is slightly different, and we adopt **Int** construction as it is suited to our purpose.

Definition 4.7 ([22]) Let $\mathbb{C}$ be a TSMC. We define a category $\mathbf{Int}(\mathbb{C})$ by the following data:

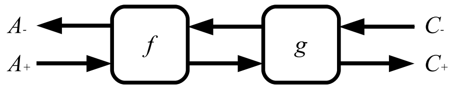
- An object is a pair (A^-, A^+) of $\mathbb{C}$-objects.
- A morphism $f : (A^-, A^+) \to (B^-, B^+)$ is a $\mathbb{C}$-morphism $f : A^+ \otimes B^- \to B^+ \otimes A^-$. The composition of f with $g : (B^-, B^+) \to (C^-, C^+)$ is defined as follows:

$$\mathbf{Tr}^{B^-}_{A^+\otimes C^-, C^+\otimes A^-}((C^+ \otimes c) \circ (g \otimes A^-) \circ (B^+ \otimes c) \circ (f \otimes C^-) \circ (A^+ \otimes c)).$$

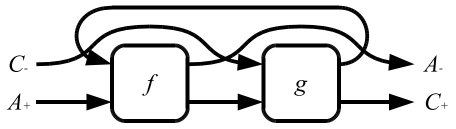
Category $\mathbf{Int}(\mathbb{C})$ is an ideal platform for representing bidirectional computation. An $\mathbf{Int}(\mathbb{C})$-morphism $f : (A^-, A^+) \to (B^-, B^+)$ stands for the bidirectional computation in Figure 3, and is "implemented" by a $\mathbb{C}$-morphism depicted as follows:



The composition of f with $g : (B^-, B^+) \to (C^-, C^+)$ is done by juxtaposition in the level of bidirectional computation:



while the above computation is realized in $\mathbb{C}$ as follows (the composition in Definition 4.7):



To understand a category-theoretic nature of **Int** construction, we introduce *compact closed categories*.

Definition 4.8 ([24]) A *compact closed category* is a symmetric monoidal category $\mathbb{C}$ such that each object A has a *left dual object* $A^\vee$ together with counit $\varepsilon_A : A^\vee \otimes A \to \mathbf{I}$ and unit $\eta_A : \mathbf{I} \to A \otimes A^\vee$ subject to certain axioms; see [24] for detail.

In fact $\mathbf{Int}(\mathbb{C})$ is a compact closed category. Here we only present object parts of the unit, tensor and left dual object in $\mathbf{Int}(\mathbb{C})$.

$$\begin{array}{rcl} \mathbf{I}_{\mathbf{Int}(\mathbb{C})} & = & (\mathbf{I}_{\mathbb{C}}, \mathbf{I}_{\mathbb{C}}) \\ (A^-, A^+) \otimes_{\mathbf{Int}(\mathbb{C})} (B^-, B^+) & = & (A^- \otimes_{\mathbb{C}} B^-, A^+ \otimes_{\mathbb{C}} B^+) \\ (A^-, A^+)^\vee & = & (A^+, A^-). \end{array}$$

Definition 4.9 1. Small TSMCs, traced symmetric monoidal functors and monoidal natural transformations form a 2-category, which we write by **TSMC**.

2. Small compact closed categories, symmetric strong monoidal functors and monoidal natural transformations form a 2-category, which we write by **CCL**.

Proposition 4.10 ([22]) *Every compact closed category has a unique trace operator; hence* **CCL** *is a 2-subcategory of* **TSMC**.

Theorem 4.11 ([22]) *The mapping $\mathbb{C} \mapsto \mathbf{Int}(\mathbb{C})$ extends to a 2-functor from* **TSMC** *to* **CCL***. Furthermore, it is a left biadjoint to the forgetful 2-functor $\mathcal{U} : \mathbf{CCL} \to \mathbf{TSMC}$. An explicit description of the unit $\mathcal{N}_{\mathbb{C}} : \mathbb{C} \to \mathbf{Int}(\mathbb{C})$ of this biadjunction is*

$$\begin{aligned} \mathcal{N}_{\mathbb{C}}(A) &= (\mathbf{I}, A) \\ \mathcal{N}_{\mathbb{C}}(f) &= f, \end{aligned}$$

and $\mathcal{N}_{\mathbb{C}}$ is full faithful and symmetric strong monoidal [2].

5 Monoidal Attribute Grammars

The SMC $\mathbf{Int}(\mathbb{C})$ arising from a TSMC $\mathbb{C}$ intrinsically models bidirectional computation. This is an ideal setting for formulating the concept of attribute grammars. In $\mathbf{Int}(\mathbb{C})$ an attribute grammar is concisely modeled as an algebra of the signature corresponding to a CFG.

We fix a CFG $G = (T, N, P, S)$.

Definition 5.1 A *monoidal attribute grammar* for G in a TSMC $\mathbb{C}$ is a Σ_G-algebra $\mathcal{A} = (A, \alpha)$ in $\mathbf{Int}(\mathbb{C})$.

This short definition accurately captures the essence of attribute grammars. First, the set of sorts of Σ_G is N; so A is nothing but an assignment of an $\mathbf{Int}(\mathbb{C})$-object to each nonterminal symbol. An object in $\mathbf{Int}(\mathbb{C})$ is a pair of $\mathbb{C}$-objects that correspond to the types of information flowing in two directions. We regard them as the types of inherited and synthesized attributes, respectively. Therefore A assigns the type of inherited and synthesized attribute to each nonterminal symbol. Below we write A^-X, A^+X for the first (=inherited attribute) and second (=synthesized attribute) component of AX. We also identify “sort” and “nonterminal symbol”.

We may also require $A^-S = \mathbf{I}_{\mathbb{C}}$, if we think that each derivation tree (i.e. a Σ_G-term of sort S) should determine a global element of the synthesized attribute of S. However, we leave this requirement optional in this paper.

Let $p \in \Sigma_G^{X_1 \cdots X_{n_p} \to X}$ be an operator. This operator corresponds to a production rule $p : X \to a_0 X_1 a_1 \cdots a_{n_p - 1} X_{n_p} a_{n_p} \in P$, so α assigns to p an $\mathbf{Int}(\mathbb{C})$-morphism

$$\alpha_p : AX_1 \otimes \cdots \otimes AX_{n_p} \to AX,$$

which is, by definition, the following $\mathbb{C}$-morphism:

$$\alpha_p : \left(\bigotimes_{k=1}^{n_p} A^+X_k \right) \otimes A^-X \to A^+X \otimes \bigotimes_{k=1}^{n_p} A^-X_k.$$

[2]This theorem is a specialization of Proposition 4.1 and Proposition 5.2 in [22].

We notice that the type of α_p is very similar to the semantic rule assigned to p in the classical attribute grammars; here tensor products are used instead of binary products (this is the reason of the name “monoidal” attribute grammar).

Computation of attributes of a derivation tree of G beginning with a nonterminal symbol $X \in N$ is then captured as the interpretation of the Σ_G-term t (of sort X) corresponding to the derivation tree by $\mathcal{A}$. This interpretation yields a global element

$$\mathcal{A}[\![t]\!] : (\mathbf{I}, \mathbf{I}) \to (A^- X, A^+ X)$$

in $\mathbf{Int}(\mathbb{C})$, which is isomorphic to a $\mathbb{C}$-morphism from $A^- X$ to $A^+ X$.

The classical attribute grammars specify computation of attributes at each production rule, but the problem is that there is no safe way to compose them so that we can compute attributes of any derivation trees. Therefore in the classical definition we tend to concentrate on the attribute grammars that do not assign cyclic computation of attributes. Such attribute grammars are called *well-formed* [25]; see the next section. On the other hand The above problem does not occur in monoidal attribute grammars because we can always safely compose semantic rules by the composition operation in $\mathbf{Int}(\mathbb{C})$, and even when cyclic computation occurs it is naturally captured by the trace operator in $\mathbb{C}$.

The rest of the paper is devoted for verifying the generality and flexibility of monoidal attribute grammars.

6 Monoidal Attribute Grammars in $\mathbf{Int}(\mathbf{Rel}^+)$

We first consider monoidal attribute grammars in $\mathbf{Int}(\mathbf{Rel}^+)$. It turns out that they precisely capture local dependency graphs of attribute grammars, which are a common tool in the analysis of the classical attribute grammars.

6.1 Category $\mathbf{Int}(\mathbf{Rel}^+)$

We first recall an explicit description of $\mathbf{Int}(\mathbf{Rel}^+)$ from [22]. A morphism $f : (A, B) \to (C, D)$ in $\mathbf{Int}(\mathbf{Rel}^+)$ is a binary relation $f : B + C \nrightarrow D + A$. This can be decomposed into four components (presented in matrix form):

$$\begin{pmatrix} f_{BD} & f_{BA} \\ f_{CD} & f_{CA} \end{pmatrix}$$

where each component f_{XY} ($X = C/B, Y = A/D$) is the subrelation of f such that f_{XY} relates elements injected from X and Y. The composition of f with another morphism $g : (C, D) \to (E, F)$ yields the morphism $g \circ f : (A, B) \to (E, F)$ given by the following binary relation of type $B + E \nrightarrow F + A$ (we let $h = g_{DC} \circ f_{CD}$ and $i = f_{CD} \circ g_{DC}$ below):

$$\begin{pmatrix} g_{DF} \circ i^* \circ f_{BD} & (f_{CA} \circ g_{DC} \circ i^* \circ f_{BD}) \cup f_{BA} \\ (g_{DF} \circ f_{CD} \circ h^* \circ g_{EC}) \cup g_{EF} & f_{CA} \circ h^* \circ g_{EC} \end{pmatrix},$$

where h^*, i^* means the transitive reflexive closure of h, i, respectively. We say that the composition of g and f *contains cycle* if h^* in the above matrix contains (d, d) for some $d \in D$ (equivalently $(c, c) \in i^*$ for some $c \in C$).

The monoidal structure over **Int**(**Rel**$^+$) is given as follows. The unit object is the pair $(\emptyset, \emptyset)$. The tensor product of two morphisms $f : (A, B) \to (C, D)$ and $g : (E, F) \to (G, H)$ is $f \otimes g : (A+E, B+F) \to (C+G, D+H)$ that is given by the following binary relation of type $B + F + C + G \nrightarrow D + H + A + E$:

$$\begin{pmatrix} f_{BD} & \emptyset & f_{BA} & \emptyset \\ \emptyset & g_{FH} & \emptyset & g_{FE} \\ f_{CD} & \emptyset & f_{CA} & \emptyset \\ \emptyset & g_{GH} & \emptyset & g_{GE} \end{pmatrix}.$$

In **Int**(**Rel**$^+$) a global element $f : (\emptyset, \emptyset) \to (A, B)$ is isomorphic to the binary relation $f_{AB} : A \nrightarrow B$ because the other three components $f_{A\emptyset}$, $f_{\emptyset B}$, $f_{\emptyset\emptyset}$ in the matrix representation of f are the empty relations (recall that $\emptyset$ is initial and terminal in **Rel**$^+$). Therefore we identify a binary relation $A \nrightarrow B$ and a global element of type (A, B).

6.2 Local Dependency Graphs and Monoidal Attribute Grammars

In the classical attribute grammars, detecting cycles in the computation of attributes is an important issue. In [25, 26] Knuth proposed an algorithm to detect the possibility of cyclic attribute computation from a local dependency graph of an attribute grammar. In this section we give a different presentation of the algorithm via the correspondence between local dependency graphs and monoidal attribute grammars in **Int**(**Rel**$^+$).

We fix a CFG $G = (T, N, P, S)$.

Definition 6.1 ([25, 10]) A *local dependency graph* D of a classical attribute grammar (**Inh**, **Syn**, V, f) for G assigns to each production rule $p : X \to a_0 X_1 a_1 \ldots X_{n_p} a_{n_p} \in P$ a directed di-graph D_p in which arcs go from an element in

$$\mathbf{Syn}\, X_1 + \ldots + \mathbf{Syn}\, X_{n_p} + \mathbf{Inh}\, X$$

to an element in

$$\mathbf{Syn}\, X + \mathbf{Inh}\, X_1 + \ldots + \mathbf{Inh}\, X_{n_p}.$$

We write ι_i and ι'_i $(0 \le i \le n_p)$ for the injection function to each disjoint union, respectively.

A local dependency graph of an attribute grammar is usually provided by hand so that it represents an intended dependency relation between inputs and outputs of each semantic rule. In principle we can not extract it from the classical attribute grammars because semantic rules, which are mere mathematical functions over some sets, do not carry such information. To resolve this problem, we may need to introduce the concept of “syntax” and “semantics” of attribute grammars. However, we do not study this issue in this paper.

Typically, we depict the di-graph D_p with **Inh** X, **Syn** X on the top and **Inh** X_i, **Syn** X_i on the bottom.

Example 6.2 ([25]) The set of di-graphs in Figure 4 gives a local dependency graph D_B of the classical attribute grammar U_B in Section 2.

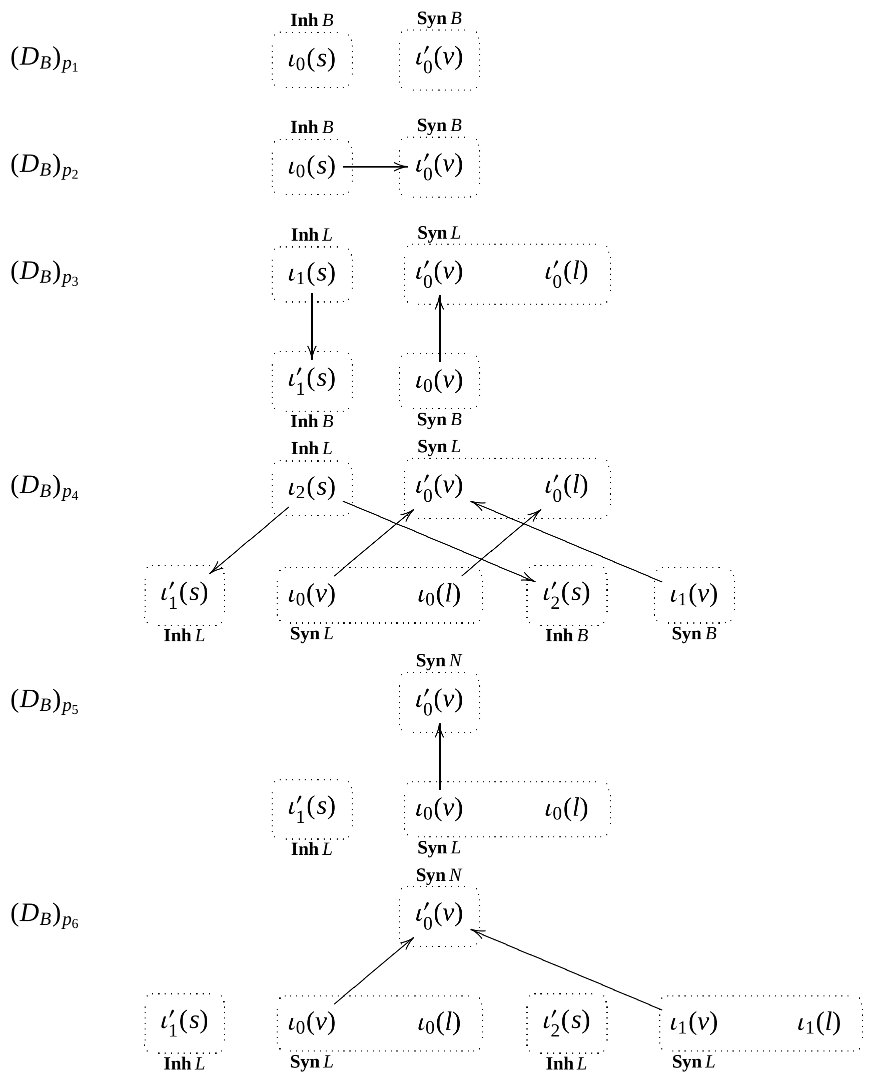
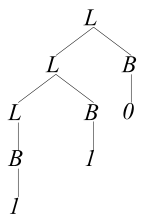


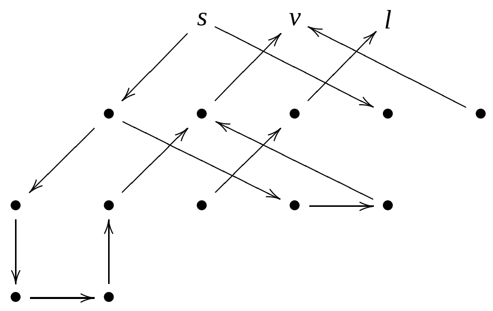
Figure 4: Local Dependency Graph D_B of U_B

The *compound dependency graph* [10] of a Σ_G-term t of a sort $X \in N$ under a local dependency graph D is the graph constructed as follows: we first paste D_p at every node of t corresponding to the production rule p, then remove the injection symbols in the top of the graph, and replace the other vertices with the bullet symbol (•).

Example 6.3 ([25]) (Continued from Example 6.2) We consider a Σ_{G_B}-term $t = p_4(p_4(p_3(p_2), p_2), p_1)$ of sort L. Its tree representation is



and its compound dependency graph under D_B in Example 6.2 is the following:



Let D be a local dependency graph of a classical attribute grammar (**Inh**, **Syn**, V, f) for G. We say that the attribute grammar is *well-formed* (under D) if for any Σ_G-term t of any sort, the compound dependency graph of t under D contains no cycle.

6.3 Local Dependency Graphs and Monoidal Attribute Grammars

We identify a local dependency graph D with the following family of binary relations:

$$D_p : \left(\sum_{k=1}^{n_p} \mathbf{Syn}\, X_k\right) + \mathbf{Inh}\, X \nrightarrow \mathbf{Syn}\, X + \sum_{k=1}^{n_p} \mathbf{Inh}\, X_k.$$

From the type of D_p, it is easy to see that a local dependency graph specifies a monoidal attribute grammar $\mathcal{R}_D = (Attr, D)$ in $\mathbf{Int}(\mathbf{Rel}^+)$, where

1. *Attr* is an N-indexed family of $\mathbf{Int}(\mathbf{Rel}^+)$-objects such that

$$\text{Attr } X = (\mathbf{Inh}\, X, \mathbf{Syn}\, X),$$

   and

2. D is the local dependency graph itself. Note that D_p exactly gives an $\mathbf{Int}(\mathbf{Rel}^+)$-morphism of type

$$D_p : \mathrm{Attr}\ X_1 \otimes \ldots \otimes \mathrm{Attr}\ X_{n_p} \to \mathrm{Attr}\ X.$$

This change of view makes it possible to understand Knuth's algorithm as computations on morphisms in $\mathbf{Int}(\mathbf{Rel}^+)$. The following operation on binary relations, which plays a central role in Knuth's dependency checking algorithm [26], can be captured as a composition in $\mathbf{Int}(\mathbf{Rel}^+)$. Let $p : X \to a_0 X_1 a_1 \cdots a_{n_p-1} X_{n_p} a_{n_p} \in P$ be a production rule and $g_i : \mathbf{Inh}(X_i) \nrightarrow \mathbf{Syn}(X_i)$ $(1 \le i \le n_p)$ be a family of binary relations. We define a binary relation $\phi(D_p, g_1, \cdots, g_{n_p}) : \mathbf{Inh}\, X \nrightarrow \mathbf{Syn}\, X$ by

$$\begin{array}{ll} & (x, y) \in \phi(D_p, g_1, \cdots, g_{n_p}) \\ \Longleftrightarrow & \text{there is a path from } \iota_{n_p}(x) \text{ to } \iota'_0(y) \text{ in } D_p[g_1, \cdots, g_{n_p}] \end{array}$$

where $D_p[g_1, \cdots, g_{n_p}]$ is the graph obtained by adding to D_p an arc from $\iota'_i(x)$ to $\iota_{i-1}(y)$ for every $1 \le i \le n_p$ and $(x, y) \in g_i$; see p.136, [25]. In fact, we have

$$\phi(D_p, g_1, \cdots, g_{n_p}) = D_p \circ (g_1 \otimes \cdots \otimes g_{n_p}) \tag{3}$$

Furthermore, the composition on the right hand side contains a cycle if and only if $D_p[g_1, \cdots, g_{n_p}]$ contains a cycle.

Equation (3) suggests that for any Σ_G-term t of a sort $X \in N$, the binary relation $\mathcal{R}_D[\![t]\!] : \mathbf{Inh}\, X \nrightarrow \mathbf{Syn}\, X$ represents the total dependency of synthesized attributes of X on inherited attributes of X in the compound dependency graph of t.

Example 6.4 (Continued from Example 6.3) Let $\mathcal{R}_{D_B}$ be the monoidal attribute grammar generated from D_B. Then $\mathcal{R}_{D_B}[\![t]\!]$ is the binary relation depicted as follows:

$$\begin{tikzcd} s \arrow[r] & v & l \end{tikzcd}$$

This relation is also obtained by collecting all the paths from $\mathbf{Inh}(L) = \{s\}$ to $\mathbf{Syn}(L) = \{v, l\}$ in the top of the compound dependency graph in Example 6.3.

Knuth's Algorithm for Detecting Circularlity

The monoidal attribute grammar $\mathcal{R}_D$ generated from a local dependency graph D gives the following set-theoretic Σ_G-algebra $\mathcal{A}_D$ via the global element monoidal functor (Definition 3.5):

$$\mathcal{A}_D = (A_D, \alpha_D) = \mathbf{Alg}_{\Sigma_G}(G_{\mathbf{Int}(\mathbf{Rel}^+)})(\mathcal{R}_D).$$

The explicit description of $\mathcal{A}_D$ is the following:

$$\begin{array}{rcll} A_D X & = & \mathbf{Int}(\mathbf{Rel}^+)((\emptyset, \emptyset), (\mathbf{Inh}\, X, \mathbf{Syn}\, X)) & \\ & \cong & \mathcal{P}(\mathbf{Inh}\, X \times \mathbf{Syn}\, X) & \\ (\alpha_D)_p(g_1, \cdots, g_{n_p}) & = & D_p \circ (g_1 \otimes \cdots \otimes g_{n_p}) & \\ & = & \phi(D_p, g_1, \cdots, g_{n_p}) & \text{(from (3))} \end{array}$$

Each carrier set of $\mathcal{A}_D$ is finite since the $\mathbf{Inh}\,X, \mathbf{Syn}\,X$ are both finite. This means that we can calculate the reachable part of $\mathcal{A}_D$ by iterating a monotonic operation over the subsets of carrier sets for a finite number of times. We define N-indexed families $\{Q_n X\}_{X \in N}$ of subsets of A_D by induction on $n \in \mathbf{N}$ as follows:

$$\begin{array}{rcl} Q_0 X & = & \emptyset \\ Q_{n+1} X & = & Q_n X \cup \\ & & \{(\alpha_D)_p(g_1, \cdots, g_{n_p}) \mid p \in \Sigma_G^{X_1 \cdots X_{n_p} \to X}, g_i \in Q_n X_i\} \end{array}$$

This increasing sequence of N-indexed families of sets will reach a fixpoint at a sufficiently large $n \in \mathbf{N}$ due to the finiteness of A_D. We write Q_∞ for the fixpoint of the above sequence. This fixpoint collects the reachable elements of $\mathcal{A}_D$.

Proposition 6.5 *For any sort $X \in N$, we have*

$$Q_\infty X = \{\mathcal{A}_D[\![t]\!] \mid t \in T_{\Sigma_G} X\}.$$

Theorem 6.6 *[26] Let D be a local dependency graph of an attribute grammar* $(\mathbf{Inh}, \mathbf{Syn}, V, f)$ *of a CFG $G = (T, N, P, S)$. Then the attribute grammar is well-formed with respect to D if and only if for each production rule $p : X \to a_0 X_1 a_1 \cdots a_{n_p - 1} X_{n_p} a_{n_p} \in P$ and $g_i \in Q_\infty X_i$ $(1 \leq i \leq n_p)$ the composition of D_p and $g_1 \otimes \cdots \otimes g_{n_p}$ contains no cycle.*

7 Monoidal Attribute Grammars in ωCPPO

In [6] Chirica and Martin proposed a domain-theoretic formulation of attribute grammars. The basic idea of the formulation is that we construct simultanious recursive equations on attributes at every node of a given derivation tree, then solve the equations by the least fixpoint operator at once. In this approach we can calculate attributes of arbitrary attribute grammars, because circular computation of attributes is acceptable in domain theory. This is in contrast to the classical attribute grammars where we have to assure that the computation of attributes have no cycles.

We fix a CFG $G = (T, N, P, S)$.

Definition 7.1 ([6]) A *K-system* for G is a tuple $\mathcal{D} = (D^-, D^+, f)$ such that

- D^-, D^+ are N-indexed families of ω-CPPOs called *inherited* and *synthesized attribute domains*, respectively. We write DX for the product $D^- X \times D^+ X$.
- f is a P-indexed family of continuous functions called *semantic function* such that f_p has the following type for each production rule $p : X \to a_0 X_1 a_1 \ldots X_{n_p} a_{n_p} \in P$:
$$f_p : \left[DX \times \left(\prod_{k=1}^{n_p} DX_k \right) \to D^+ X \times \prod_{k=1}^{n_p} D^- X_k \right].$$

The synthesized attribute of a Σ_G-term t of a sort $X \in N$ is calculated from an inherited attribute $i \in D^- X$ as follows. We first construct the following product domain by induction on the structure of t:

$$D^{p(t_1,\dots,t_{n_p})} = D^+X \times \left(\prod_{k=1}^{n_p} D^-X_k\right) \times \left(\prod_{k=1}^{n_p} D^{t_k}\right) \qquad (p \in \Sigma_G^{X_1\dots X_{n_p}\to X}).$$

For $d \in D^t$, by $\mathrm{fst}(d)$ we mean the first component of d (so $\mathrm{fst}(d) \in D^+X$). Next, we construct a continuous function $H^t : [D^-X \times D^t \to D^t]$ by induction on the structure of t.

$$\begin{array}{rl} & H^{p(t_1,\dots,t_{n_p})}(i, s, i_1, \dots, i_{n_p}, w_1, \dots, w_{n_p}) \\ = & (f_p(i, s, i_1, \mathrm{fst}(w_1), \dots, i_{n_p}, \mathrm{fst}(w_{n_p})), H^{t_1}(i_1, w_1), \dots, H^{t_{n_p}}(i_{n_p}, w_{n_p})). \end{array}$$

This function congregates one-step computation of inherited and synthesized attributes at *every node* of t. We apply the least fixpoint operator to $H^t(i, -) : [D^t \to D^t]$, and obtain the synthesized attribute by the first projection. We summarize this process by the following function $\mathcal{D}(t) : [D^-X \to D^+X]$:

$$\mathcal{D}(t)(i) \quad = \quad \mathrm{fst}(\mathbf{fix}(\lambda x.H^t(i, x))).$$

7.1 Correspondence between K-systems and Monoidal Attribute Grammars

We can obtain a monoidal attribute grammar in ω**CPPO** from a K-system by feeding-back the outputs of each semantic function to its inputs. On the other hand every monoidal attribute grammar in ω**CPPO** can be casted to a K-system in an obvious way. We show that these constructions preserve meanings of Σ_G-terms. In this sense K-systems and monoidal attribute grammars in ω**CPPO** are equivalent.

We fix a CFG $G = (T, N, P, S)$.

Definition 7.2 Let $\mathcal{D} = (D^-, D^+, f)$ be a K-system for G. From this K-system we construct a monoidal attribute grammar $M(\mathcal{D}) = (D, \delta)$ for G in ω**CPPO** where

- D is a N-indexed family of **Int**(ω**CPPO**)-objects such that $DX = (D^-X, D^+X)$ for each $X \in N$.

- δ is a P-indexed family of **Int**(ω**CPPO**)-morphisms such that for each operator $p \in \Sigma_G^{X_1\cdots X_{n_p}\to X}$,

$$\delta_p : \left[\left(\prod_{k=1}^{n_p} D^+X_k\right) \times D^-X \to D^+X \times \prod_{k=1}^{n_p} D^-X_k\right]$$

  is a continuous function defined by

$$\delta_p(s_1, \dots, s_{n_p}, i) = \mathbf{fix}(\lambda(s, i_1, \dots, i_{n_p})\ .\ f_p(i, s, i_1, s_1, \dots, i_{n_p}, s_{n_p})).$$

Conversely any monoidal attribute grammar $\mathcal{A}$ determines a K-system $K(\mathcal{A})$ in an obvious way.

Below we identify an **Int**(ω**CPPO**)-morphism $f : (1, 1) \to (X, Y)$ and a continuous function $f : [X \to Y]$.

Theorem 7.3 *Let t be a Σ_G-term of a sort $X \in N$. Then for any K-system $\mathcal{D}$ for G and monoidal attribute grammar $\mathcal{A}$ for G in ω**CPPO**, we have*

1. *$M(\mathcal{D})\,[\![t]\!] = \mathcal{D}(t)$, and*
2. *$K(\mathcal{A})(t) = \mathcal{A}\,[\![t]\!]$.*

Proof Easy induction on the structure of t.

The above discussion also suggests that in TCCs we can model *full* attribute grammars, where the semantic rule for a production rule $p : X \to a_0X_1 \dots a_{n-1}X_{n_p}a_{n_p} \in P$ is given by a morphism of the following type:

$$f_p : DX \times \left(\prod_{k=1}^{n_p} DX_k\right) \to D^+X \times \prod_{k=1}^{n_p} D^-X_k.$$

In TCCs trace operators specify Conway operators (Theorem 4.6). Therefore by taking the fixpoint of

$$f_p \circ \alpha : \left(\left(\prod_{k=1}^{n_p} D^+X_k\right) \times D^-X\right) \times \left(D^+X \times \prod_{k=1}^{n_p} D^-X_k\right) \to D^+X \times \prod_{k=1}^{n_p} D^-X_k,$$

(α is an appropriate isomorphism) we obtain monoidal attribute grammars in TCCs.

8 Equivalence between Monoidal Attribute Grammars and Synthesized Ones

Attribute grammars that do not use inherited attributes are called *synthesized attribute grammar* (*S-attribute grammar* for short). In [6] Chirica and Martin observed that by using function spaces as attribute domains every K-system can be reduced to the one which does not use inherited attributes. Based on the same observation, in [12] the authors showed that every attribute grammars can be represented by higher-order catamorphisms.

In this section we show a similar result for monoidal attribute grammars. We introduce a monoidal version of synthesized attribute grammars (*monoidal S-attribute grammars*), and show that in traced symmetric monoidal *closed* categories every monoidal attribute grammar is equivalent to a monoidal S-attribute grammar. We fix a CFG $G = (T, N, P, S)$.

Definition 8.1 A *monoidal S-attribute grammar* for G in a TSMC $\mathbb{C}$ is a monoidal attribute grammar (A, α) for G such that $A^-X = \mathbf{I}$ (equivalently $AX = \mathcal{N}_{\mathbb{C}}(A^+X)$) for each sort $X \in N$.

Every Σ_G-algebra in $\mathbb{C}$ carries the same information as a monoidal S-attribute grammar.

Proposition 8.2 *For any TSMC $\mathbb{C}$, the functor*

$$\mathbf{Alg}_{\Sigma_G}(\mathcal{N}_{\mathbb{C}}) : \mathbf{Alg}_{\Sigma_G}(\mathbb{C}) \to \mathbf{Alg}_{\Sigma_G}(\mathbf{Int}(\mathbb{C}))$$

can be restricted to the equivalence between $\mathbf{Alg}_{\Sigma_G}(\mathbb{C})$ and the full subcategory of $\mathbf{Alg}_{\Sigma_G}(\mathbf{Int}(\mathbb{C}))$ specified by monoidal S-attribute grammars.

We show that the computational content specified by a Σ_G-algebra in $\mathbb{C}$ is the same as the one specified by the corresponding monoidal S-attribute grammar. We capture this claim by the equivalence of algebras (Definition 3.7).

Lemma 8.3 *The monoidal natural transformation $G_{\mathcal{N}_{\mathbb{C}}} : G_{\mathbb{C}} \to G_{\mathbf{Int}(\mathbb{C})} \circ \mathcal{N}_{\mathbb{C}}$ is a monoidal natural isomorphism.*

Proof The isomorphism is given as follows:

$$\begin{array}{rcll} G_{\mathbb{C}}(C) & = & \mathbb{C}(\mathbf{I}, C) & \\ & \cong & \mathbf{Int}(\mathbb{C})(\mathcal{N}_{\mathbb{C}}\mathbf{I}, \mathcal{N}_{\mathbb{C}}C) & (\mathcal{N}_{\mathbb{C}} \text{ is full faithful}) \\ & = & \mathbf{Int}(\mathbb{C})(\mathbf{I}, \mathcal{N}_{\mathbb{C}}C) & \\ & = & G_{\mathbf{Int}(\mathbb{C})} \circ \mathcal{N}_{\mathbb{C}}(C). & \end{array}$$

Theorem 8.4 *For any TSMC $\mathbb{C}$, every Σ_G-algebra $\mathcal{A}$ in $\mathbb{C}$ is equivalent to a monoidal S-attribute grammar $\mathbf{Alg}_{\Sigma_G}(\mathcal{N}_{\mathbb{C}})(\mathcal{A})$ in $\mathbb{C}$.*

Proof From 2-functoriality of $\mathbf{Alg}_{\Sigma_G}$, monoidal natural isomorphism $G_{\mathcal{N}_{\mathbb{C}}}$ in Lemma 8.3 gives the isomorphism $\mathbf{Alg}_{\Sigma_G}(G_{\mathcal{N}_{\mathbb{C}}})$ between $\mathcal{A}$ and $\mathbf{Alg}_{\Sigma_G}(\mathcal{N}_{\mathbb{C}})(\mathcal{A})$.

To show that *every* monoidal attribute grammar is equivalent to a monoidal S-attribute grammar, we need an extra structure on $\mathbb{C}$. We assume that $\mathbb{C}$ is *closed*, that is, $- \otimes B$ has a right adjoint $B \multimap -$ for every object B in $\mathbb{C}$. The key to prove the equivalence is the following theorem due to Hasegawa.

Theorem 8.5 ([19]) *Let $\mathbb{C}$ be a TSMC. Then $\mathcal{N}_{\mathbb{C}} : \mathbb{C} \to \mathbf{Int}(\mathbb{C})$ has a symmetric monoidal right adjoint if and only if $\mathbb{C}$ is closed.*

Lemma 8.6 *Let $\mathbb{C}$ be a traced symmetric monoidal closed category. We write $\mathcal{R}_{\mathbb{C}} : \mathbf{Int}(\mathbb{C}) \to \mathbb{C}$ for the symmetric monoidal right adjoint that exists by Theorem 8.5.*

The monoidal natural transformation $G_{\mathcal{R}_{\mathbb{C}}} : G_{\mathbf{Int}(\mathbb{C})} \to G_{\mathbb{C}} \circ \mathcal{R}_{\mathbb{C}}$ is a monoidal natural isomorphism.

Proof We first consider the following natural isomorphism:

$$\begin{array}{rcl} G_{\mathrm{Int}(\mathbb{C})} & = & \mathrm{Int}(\mathbb{C})(\mathbf{I}, -) \\ & = & \mathrm{Int}(\mathbb{C})(\mathcal{N}_{\mathbb{C}}\mathbf{I}, -) \\ & \cong & \mathbb{C}(\mathbf{I}, \mathcal{R}_{\mathbb{C}}-) \\ & = & G_{\mathbb{C}} \circ \mathcal{R}_{\mathbb{C}}. \end{array}$$

This isomorphism maps a global element $f \in G_{\mathbf{Int}(\mathbb{C})}(A)$ to $\mathcal{R}_{\mathbb{C}}(f) \circ \mathcal{R}_{\mathbb{C}}((m^{\mathcal{N}_{\mathbb{C}}})^{-1}) \circ \eta_{\mathbf{I}}$. Recall that $\mathcal{N}_{\mathbb{C}}$ is strong monoidal; so $m^{\mathcal{N}_{\mathbb{C}}} : \mathbf{I}_{\mathbf{Int}(\mathbb{C})} \to \mathcal{N}_{\mathbb{C}}\mathbf{I}_{\mathbb{C}}$ is an isomorphism. Since $\mathcal{N}_{\mathbb{C}} \dashv \mathcal{R}_{\mathbb{C}}$ is a monoidal adjunction, we have $(m^{\mathcal{N}_{\mathbb{C}}})^{-1} = \epsilon_{\mathbf{I}} \circ \mathcal{N}_{\mathbb{C}}(m^{\mathcal{R}_{\mathbb{C}}})$. Therefore the passage from the top to bottom is equal to $G_{\mathcal{R}_{\mathbb{C}}}$.

Theorem 8.7 *For any traced symmetric monoidal closed category $\mathbb{C}$, every Σ_G-algebra $\mathcal{A}$ in* $\mathbf{Int}(\mathbb{C})$ *is equivalent to* $\mathbf{Alg}_{\Sigma_G}(\mathcal{R}_{\mathbb{C}})(\mathcal{A})$ *in $\mathbb{C}$.*

Proof Similar to the proof of Theorem 8.4.

Corollary 8.8 *In any traced symmetric monoidal closed category $\mathbb{C}$, every monoidal attribute grammar for G is equivalent to a monoidal S-attribute grammar for G.*

Category $\omega\mathbf{CPPO}$ is a traced Cartesian closed category, and Theorem 4.4 and Section 4.5 in [6] is an instance of Theorem 8.7 and Corollary 8.8 with $\mathbb{C} = \omega\mathbf{CPPO}$.

9 Related Work

In [25], Knuth proposed attribute grammars as a formalism to interpret derivation trees of context free grammars. Since then attribute grammars has been extensively studied both from theoretical and practical perspective.

There are many works on the classification of well-formed attribute grammars; there is a good survey in [10]. In this paper we do not regard cyclic computation of attributes as a bad phenomenon; we rather aim to establish a categorical formulation of general attribute grammars including non-well formed ones.

The primary application of attribute grammars is compiler description languages. In this area various extensions of the (classical) attribute grammars have been proposed to make languages flexible and strong [11, 5, 4, 29]. We do not know any formal relationship between monoidal attribute grammars and these extensions. To catch up with these advanced attribute grammars further investigation is needed.

Recently, attribute grammars are recasted to the foundation for fusion transformations in functional languages [7, 8, 30, 28, 23]. These works are influenced by Ganzinger and Giegerich's descriptional composition [13, 14, 15]. We believe that our categorical formulation of attribute grammars provides a new perspective on program transformations.

In this paper we studied monoidal attribute grammars within two concrete TSMCs, namely $\mathbf{Rel}^+$ and $\omega\mathbf{CPPO}$. In fact these categories are TCCs, and we have not yet seen any example using proper TSMCs. The author is currently working on a categorical formulation of the descriptional composition, where the full generality of monoidal attribute grammars and the universal property of **Int** construction (Theorem 4.11) will be exploited.

Girard proposed a new paradigm, called *geometry of interaction* (GoI), for modeling dynamics of cut-elimination procedure in linear logic [16, 17, 18]. Girard's models were later analyzed and axiomatized in subsequent studies [9, 1, 3, 2]. The concept that contributed to the analysis of GoI program is *traced monoidal categories*, which were introduced in the context of the mathematical models of knots and tangles [22].

The connection between trace operators and parameterized fixpoint operators in Cartesian categories is discovered independently by Hasegawa [20] and Hyland. This connection allows us to consider attribute grammars in various domain-theoretic categories. Chirica and Martin's work can be seen as a practice of this observation.

10 Acknowledgements

The author is grateful to Masahito Hasegawa for technical advises. The author is also grateful to Susumu Nishimura for many discussions on attribute grammars and program transformations. I thank Kazuyuki Asada and Naohiko Hoshino for their comments.

References

[1] Samson Abramsky. Retracting some paths in process algebra. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1996.

[2] Samson Abramsky, Esfandiar Haghverdi, and Philip J. Scott. Geometry of interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002.

[3] Samson Abramsky and Radha Jagadeesan. New foundations for the geometry of interaction. *Inf. Comput.*, 111(1):53–119, 1994.

[4] John Boyland. Conditional attribute grammars. *ACM Trans. Program. Lang. Syst.*, 18(1):73–108, 1996.

[5] John Boyland and Susan L. Graham. Composing tree attributions. In *POPL '94*, pages 375–388, 1994.

[6] Laurian M. Chirica and David F. Martin. An order-algebraic definition of knuthian semantics. *Mathematical Systems Theory*, 13:1–27, 1979.

[7] Loïc Correnson, Etienne Duris, Didier Parigot, and Gilles Roussel. Symbolic composition. Technical Report 3348, INRIA, January 1998.

[8] Loïc Correnson, Etienne Duris, Didier Parigot, and Gilles Roussel. Declarative program transformation: A deforestation case-study. In *PPDP '99*, volume 1702 of *LNCS*, pages 360–377. Springer Verlag, 1999.

[9] Vincent Danos and Laurent Regnier. Local and asynchronous beta-reduction (an analysis of girard's execution formula). In *LICS '93*, pages 296–306. IEEE Computer Society, 1993.

[10] Pierre Deransart, Martin Jourdan, and Bernard Lorho. *Attribute Grammars; Definitions, Systems and Bibliography*, volume 323 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988.

[11] Rodney Farrow, Thomas J. Marlowe, and Daniel M. Yellin. Composable attribute grammars: Support for modularity in translator design and implementation. In *POPL '92*, pages 223–234, 1992.

[12] Maarten Fokkinga, Johan Jeuring, Lambert Meertens, and Erik Meijer. A translation from attribute grammars to catamorphisms. *The Squiggolist*, 2(1):20–26, 1991.

[13] Harald Ganzinger. Increasing modularity and language-independency in automatically generated compilers. *Sci. Comput. Program.*, 3(3):223–278, 1983.

[14] Harald Ganzinger and Robert Giegerich. Attribute coupled grammars. In *SIGPLAN Symposium on Compiler Construction '84*, pages 157–170. ACM, 1984.

[15] Robert Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Inf.*, 25(4):355–423, 1988.

[16] Jean-Yves Girard. Geometry of Interaction I: Interpretation of System F. In R. Ferro et al., editor, *Logic Colloquium '88*. North-Holland, 1989.

[17] Jean-Yves Girard. Geometry of Interaction II: Deadlock-free Algorithms. In Per Martin-Löf and Grigori Mints, editors, *Conference on Computer Logic '88*, volume 417 of *Lecture Notes in Computer Science*, pages 76–93. Springer, 1990.

[18] Jean-Yves Girard. Geometry of Interaction III: Accommodating the Additives. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, number 222 in London Math. Soc. Series. Cambridge University Press, 1995.

[19] Masahito Hasegawa. On traced monoidal closed categories. Invited Talk in Traced Monoidal Categories, Network Algebras, and Applications 2007.

[20] Masahito Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of* let *and* letrec. Springer-Verlag, 1999.

[21] Andre Joyal and Ross Street. The geometry of tensor calculus I. *Adv. Math.*, 88:55–112, 1991.

[22] Andre Joyal, Ross Street, and Dominc Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.

[23] Shinya 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.

[24] G. M. Kelly and M. L. Laplaza. Coherence for compact closed categories. *Journal of Pure and Applied Algebra*, 19:193–213, 1980.

[25] Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

[26] Donald E. Knuth. Correction: Semantics of context-free languages. *Mathematical Systems Theory*, 5(1):95–96, 1971.

[27] Saunders MacLane. *Categories for the Working Mathematician (Second Edition)*, volume 5 of *Graduate Texts in Mathematics*. Springer, 1998.

[28] Susumu Nishimura. Deforesting in accumulating parameters via type-directed transformations. In *APLAS '02*, pages 145–159, 2002.

[29] S. Doaitse Swierstra and Harald Vogt. Higher order attribute grammars. In Henk Alblas and Borivoj Melichar, editors, *Attribute Grammars, Applications and Systems*, volume 545 of *Lecture Notes in Computer Science*, pages 256–296. Springer, 1991.

[30] Janis Voigtländer. Using circular programs to deforest in accumulating parameters. *Higher-Order and Symbolic Computation*, 17(1-2):129–163, 2004.