Math. Struct. in Comp. Science (2013), *vol.* 23, *pp.* 204–246. © Cambridge University Press 2013
doi:10.1017/S0960129512000096

The monoidal structure of Turing machines[†]

MIKLÓS BARTHA

Department of Computer Science, Memorial University of Newfoundland
St. John's, NL, Canada
Email: bartha@inf.u-szeged.hu

Received 15 October 2010; revised 23 August 2011

Indexed monoidal algebras are introduced as an equivalent structure for self-dual compact closed categories, and a coherence theorem is proved for the category of such algebras. Turing automata and Turing graph machines are defined by generalising the classical Turing machine concept so that the collection of such machines becomes an indexed monoidal algebra. On the analogy of the von Neumann data-flow computer architecture, Turing graph machines are proposed as potentially reversible low-level universal computational devices, and a truly reversible molecular size hardware model is presented as an example.

1. Introduction

The importance of reversibility in computation has been argued at several levels in connection with the speed and efficiency of modern-day computers. As stated originally by Landauer (Landauer 1961) and re-emphasised by Abramsky (Abramsky 2005a), 'it is only the logically irreversible operations in a physical computer that necessarily dissipate energy by generating a corresponding amount of entropy for every bit of information that gets irreversibly erased'. Abramsky's remedy for this situation in Abramsky (2005a) is to translate high-level functional programs in a syntax-directed way into a simple kind of automata that are immediately seen to be reversible. The concept of a strong compact closed category (Abramsky 2005b) has been introduced and advocated as a theoretical foundation for this type of reversibility.

The problem of reversibility, however, does not manifest itself at the software level. Even if we manage to perform our programs in reverse, it is not guaranteed that information will not be lost during the concrete physical computation process. Indeed, it may get lost twice: once in each direction. The solution must therefore be found at the lowest hardware level. Our model of Turing graph machines is being presented as a possible hardware solution for the problem of reversibility, but follows Abramsky's structural approach. We even go one step further by showing how computations can be done in a virtually undirected fashion under the theoretical umbrella of self-dual compact closed categories. In practical terms, we mean that, unlike in synchronous systems (for example, sequential circuits), where the information is always propagated in the same direction through the interconnections (wires) between the functional elements (logical gates), in a Turing graph machine the flow of information along these interconnections takes a direction that is

[†] This work was partially supported by the Natural Science and Engineering Research Council of Canada.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

determined dynamically by the current input and state of the machine. We are going to reconsider self-dual compact closed categories as indexed monoidal algebras and prove a coherence theorem to establish undirected graphs – constituting the basic underlying structure for Turing graph machines – as free indexed monoidal algebras generated by the ranked alphabet consisting of the star graphs.

Some parts of this paper need not be read in a strict sequential order. An in-depth knowledge of algebra and category theory is only required for Sections 2, 3 and 4. However, anyone less familiar with categories should still be able to understand the concept of Turing automata and Turing graph machines in Section 6, and appreciate the main practical contribution of this study. Then, gradually moving backwards to previous sections, it should be possible to understand the categorical background through the numerous examples and diagrams provided in the text to help intuition.

Our work relates to several significant theoretical endeavours to capture the geometry of the tensor and trace (feedback, iteration) operations in different models. The most important of these are linear logic (Girard 1987), game semantics (Hyland 1997), the semantics of quantum protocols and programming languages (Abramsky and Coecke 2004; D'Hondt and Panangaden 2006), communicating concurrent processes (Milner 2009), interaction nets (Lafont 1989) and Girard's Geometry of Interaction program (Girard 1989; Girard 1990) in general. The future generalisation of our coherence result to ribbon categories (balanced monoidal categories (Joyal and Street 1991)) will also make a connection to knot theory (Freyd and Yetter 1992), and other directions specified in Joyal *et al.* (1996) and Joyal and Street (1991).

However, the idea of Turing automata and graph machines was inspired solely by C. C. Elgot's work on flowchart schemes and their semantics (Elgot 1975), which was later developed by S. L. Bloom and Z. Ésik into a theory highlighted by the concept of *iteration theories* (Bloom and Ésik 1993). In a sense, the current paper provides a synthesis of finite automata and Turing machines in the framework of category theory, and introduces an indexing mechanism (Burstall *et al.* 1989) for iteration theories.

This paper is an extension of a talk given by the author at DCM 2010, Edinburgh.

2. Traced monoidal and compact closed categories

In this section we shall assume familiarity with the concept of symmetric monoidal categories (Mac Lane 1971). It is known (*cf.* Mac Lane and Paré (1985)) that every monoidal category is equivalent to a strict one. Repeating an argument from Joyal and Street (1991), most results obtained from the hypothesis that a monoidal category is strict can, in principle, be reformulated and proved without that condition. Even though this is also true for our results in the current paper, we shall not assume that our monoidal categories are strict unless this assumption is clearly technical and simplifies the discussion significantly. Since we are introducing the concept of indexed monoidal algebras in the current paper, it is appropriate that its definition be given under the general non-strict conditions. Another reason is that the principal example of Turing automata presented in Section 6 is not strict as an indexed monoidal algebra or traced monoidal category. A further generalisation of this concept for braidings, rather than symmetries, will follow.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

Thus, a *monoidal category* consists of a category $\mathscr{C}$, a bifunctor $\otimes : \mathscr{C} \times \mathscr{C} \to \mathscr{C}$, a unit object I of $\mathscr{C}$, and natural isomorphisms

$$a_{X,Y,Z} : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$$

as *associators*,

$$l_X : I \otimes X \to X$$

and

$$r_X : X \otimes I \to X$$

as left and right *unitors*, and

$$c_{U,V} : U \otimes V \to V \otimes U$$

as *symmetries*, subject to the well-known coherence axioms specified in Mac Lane (1971).

Traced monoidal categories (with one additional axiom) and their graphical language first appeared in Bartha (1987a) in an algebraic setting, where we used the name 'scheme algebra' for these structures, and the trace operation was called feedback. However, essentially the same axiomatisation and graphical language was published in Căzănescu and Ştefănescu (1990) under the name 'biflow'. Joyal, Street and Verity then rediscovered this notion, generalised it to balanced monoidal categories, and presented the fundamental *Int* construction on the embedding of an arbitrary balanced traced monoidal category into a tortile one (Joyal *et al.* 1996). When braiding is symmetry, as it is in our present study, the *Int* construction transforms an arbitrary traced monoidal category into a compact closed category.

The following definition of traced monoidal categories uses the terminology of Joyal *et al.* (1996). Trace in a monoidal category $\mathscr{C}$ is introduced as the *left* trace, that is, an operation $\mathscr{C}(U + A, U + B) \to \mathscr{C}(A, B)$, rather than $\mathscr{C}(A + U, B + U) \to \mathscr{C}(A, B)$ (that is, right trace) as it appears in Joyal *et al.* (1996), to be in accordance with the author's previous work. Let $\mathscr{C}$ be a monoidal category with tensor $\otimes$ and unit object I.

Definition 2.1. A *trace* for a symmetric monoidal category $\mathscr{C}$ is a natural family of functions

$$Tr^U_{A,B} : \mathscr{C}(U \otimes A, U \otimes B) \to \mathscr{C}(A, B)$$

satisfying the following axioms:

— *Vanishing:*

$$Tr^I_{A,B}(f) = f \quad \text{for } f : A \to B$$
$$Tr^{U \otimes V}_{A,B}(a_{U,V,A} \circ g) = Tr^V_{A,B}(Tr^U_{V \otimes A, V \otimes B}(g \circ a_{U,V,B}))$$

where $g : U \otimes (V \otimes A) \to (U \otimes V) \otimes B$;

— *Superposing:*

$$Tr^U_{A,B}(f) \otimes g = Tr^U_{A \otimes C, B \otimes D}(a^{-1}_{U,A,C} \circ (f \otimes g) \circ a_{U,B,D}),$$

where $f : U \otimes A \to U \otimes B$, and $g : C \to D$;

— *Yanking:*

$$Tr^U_{U,U}(c_{U,U}) = 1_U.$$

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

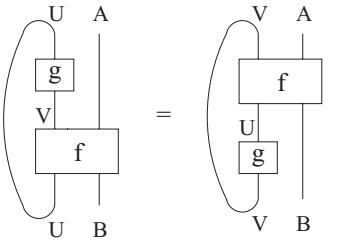


Fig. 1. The sliding axiom

Naturality of trace here means naturality in all three variables A, B, U. Naturality in A and B is self-explanatory, while (di-)naturality in U is expressed by the following separate axiom.

— *Sliding:*

$$Tr^U_{A,B}((g \otimes 1_A) \circ f) = Tr^V_{A,B}(f \circ (g \otimes 1_B)) \text{ for } f : V \otimes A \to U \otimes B,\ g : U \to V.$$

See Figure 1, and notice that we write the composition of morphisms ($\circ$) in a left-to-right manner. When using the term feedback for trace, the notation Tr changes to $\uparrow$.

Following along a separate line of development, compact closed categories were introduced in Kelly and Laplaza (1980).

Definition 2.2. A symmetric monoidal category $\mathscr{C}$ is *compact closed* (CC, for short) if every object A has a left adjoint A^* in the sense that there exist morphisms $d_A : I \to A \otimes A^*$ (the unit map) and $e_A : A^* \otimes A \to I$ (the counit map) for which the two composites below result in the identity morphisms 1_A and 1_{A^*}, respectively.

$$A \simeq_l I \otimes A \to_{d_A \otimes 1_A} (A \otimes A^*) \otimes A \simeq_a A \otimes (A^* \otimes A) \to_{1_A \otimes e_A} A \otimes I \simeq_r A,$$

$$A^* \simeq_l A^* \otimes I \to_{1_{A^*} \otimes d_A} A^* \otimes (A \otimes A^*) \simeq_a (A^* \otimes A) \otimes A^* \to_{e_A \otimes 1_{A^*}} I \otimes A^* \simeq_r A^*,$$

where l, r and a stand for appropriate left unitor, right unitor and associator morphisms, respectively.

By virtue of the adjunctions $A \dashv A^*$, there is a natural isomorphism between the hom-sets $\mathscr{C}(B \otimes A, C)$ and $\mathscr{C}(B, C \otimes A^*)$ for all objects B, C, hence the name 'compact closed' category. Category $\mathscr{C}$ is *self-dual* compact closed (SDCC, for short) if $A = A^*$ for each object A. The category **SDCC** has all SDCC categories as objects, and strict monoidal functors preserving the given self-adjunctions as morphisms.

Every CC category admits a so-called *canonical trace* (Joyal *et al.* 1996) defined by the formula

$$Tr^U_{A,B} f = (d_{U^*} \otimes 1_A) \circ (1_{U^*} \otimes f) \circ (e_U \otimes 1_B)$$

in the strict setting – see Figure 2. A well-known SDCC category is the category $(\mathbf{Rel}, \times)$ of sets and relations, with the tensor being the cartesian product $\times$. We shall use this category as an example to explain the idea of indexing on it.

According to Burstall *et al.* (1989) and Bartha and Jürgensen (1989), an *indexed family of sets* is a functor $\mathscr{I} : \mathbf{Ind} \to \mathbf{Set}$, where **Ind** is the index category. In our example, **Ind**

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

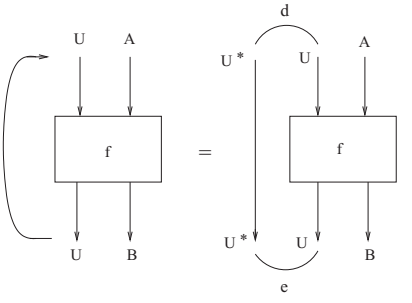


Fig. 2. Canonical trace in CC categories.

is the monoidal category (**Set**, $\times$) as a subcategory of (**Rel**, $\times$) and $\mathscr{I}$ is the covariant powerset functor $\mathscr{P}$, which is of course not monoidal. However, relations $A \to B$ are still subsets of $A \times B$, and as such they can be indexed by morphisms (functions) $A \times B \to C$ in **Set**. For any two objects (sets), we can then consider the binary operation tensor,

$$\otimes_{A,B}(= \times) : \mathscr{P}(A) \times \mathscr{P}(B) \to \mathscr{P}(A \times B),$$

and the unary operation trace,

$$\updownarrow_{A,B} : \mathscr{P}(A \times A \times B) \to \mathscr{P}(B)$$

for which

$$b \in \updownarrow_{A,B} R \quad \text{iff} \quad \exists a \in A \; ((a, a), b) \in R.$$

The concept of an indexed monoidal algebra arises from observing the equational algebraic laws satisfied by these operations and their relationship to indexing. It will soon become clear that the word 'trace' is not being abused in this context.

3. Association and permutation symbols

Following up on our SDCC category (**Rel**, $\times$) example as an indexed family of sets equipped with the operations $\otimes$ and $\updownarrow$, observe that the index category **Ind** can be chosen in many different meaningful ways without affecting the outcome of indexing, as long as **Ind** remains a sub-(monoidal) category of (**Rel**, $\times$) containing all permutations. For example, **Ind** could be chosen as the restriction of (**Set**, $\times$) to bijections, or, at the other extreme, it could even be the whole category (**Rel**, $\times$). In any case, the index functor $\mathscr{I}$ is essentially an appropriate restriction of the covariant hom-functor $Hom(I, _)$ in (**Rel**, $\times$). Clearly, the larger the index category, the less is hidden from the SDCC category to be recaptured, that is, the less interesting the whole description. Therefore, we take a minimalistic approach with regard to indexing, and abolish the index category altogether. Instead, we introduce permutation symbols, which can be interpreted as permutations in any suitable choice of a hypothetical index monoidal category. The problem is that the symbols themselves do not form a category over the given objects; in our example, sets. By Mac Lane's coherence theorem for monoidal categories, permutation symbols form a category over object terms as objects, and these terms are built up freely from the

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

given objects as object variables. Eventually, permutation symbols will be used as unary operations in the algebraic structure already containing $\otimes$ and $\updownarrow$.

3.1. *Association symbols*

Let $\tau = (\otimes, I)$ be the algebra type – this will be fixed for the rest of the paper – consisting of a binary operation symbol $\otimes$ (tensor) and a constant symbol I (unit). Furthermore, let $X = \{x_1, x_2, \ldots, x_n, \ldots\}$ be a countably infinite set of variable symbols, and $X_n = \{x_1, \ldots, x_n\}$ for each $n \in N$. The carrier set of the free τ-algebra generated by X is denoted by $T_\tau(X)$, as usual. This set consists of all τ-terms (trees) over the variables X. For a tree $t \in T_\tau(X)$, $yield(t) \in X^*$ is the string of variables appearing on the leaves of t in left-to-right order. The set $\tilde{T}_\tau(X) \subseteq T_\tau(X)$ consists of those trees t for which $yield(t) = x_1 x_2 \ldots x_n$ for some $n \in N$. The number n is called the *length* of t, denoted $l(t)$. Observe that a tree $t \in \tilde{T}_\tau(X)$ is just an alternative representation of a binary word in Mac Lane's coherence theorem (Mac Lane 1971), so every occurrence of a variable is an occurrence of the place holder symbol '_'. Note that even though we have infinitely many variables, there is only one place holder in t, since x_i always marks the ith occurrence of the place holder from left to right. In the concrete representation of trees, we shall adopt the usual prefix notation for the operation $\otimes$. For example,

$$t = \otimes(x_1, \otimes(\otimes(I, x_2), \otimes(x_3, I)))$$

is a tree in $\tilde{T}_\tau(X)$ of length 3.

We will now redefine the operation $\otimes$ over $\tilde{T}_\tau(X)$ in such a way that $t_1 \otimes t_2$ is obtained from $\otimes(t_1, t_2)$ by incrementing the index of each variable occurring in the subtree t_2 by $l(t_1)$. Notice that the notation for $\otimes$ changes from prefix to infix in the resulting τ-algebra. Clearly,

$$l(t_1 \otimes t_2) = l(t_1) + l(t_2).$$

According to Mac Lane's coherence theorem, $(\tilde{T}_\tau(X), \otimes, I)$ then becomes the object structure of a monoidal category Λ in which there is a single morphism between t and t' if and only if $l(t) = l(t')$. This is the free single-sorted monoidal category, that is, the one freely generated by a single object, which in our representation is the tree x_1.

Proposition 3.1 (Mac Lane 1971, Theorem VII.1). For every monoidal category $\mathscr{C}$ and any object C of $\mathscr{C}$, there exists a unique strict monoidal functor $\Lambda \to \mathscr{C}$ sending x_1 to C.

To switch from single-sorted to many-sorted, let O be a class of *object variables*. We define $T_\tau(O)$ as the collection of pairs (t, α), where $t \in \tilde{T}_\tau(X)$ is a tree of length n and $\alpha : X_n \to O$ is a mapping. The pair (t, α) is called an *object term*. The mapping α defines a sequence of objects $(A_1, \ldots, A_n)$ (also denoted by α), so we can identify the pair (t, α) with a single tree u, the leaves of which are labelled by the object variables in O rather than the symbols in X. Roughly speaking, O takes over the role of X, except that O is not necessarily a set.

For pairs (trees) $u_1 = (t_1, \alpha_1)$ and $u_2 = (t_2, \alpha_2)$, let

$$u_1 \otimes u_2 = (t_1 \otimes t_2, \alpha_1 \otimes \alpha_2),$$

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

where

$$(\alpha_1 \otimes \alpha_2)(x_i) = \begin{cases} \alpha_1(x_i) & \text{if } 1 \leqslant i \leqslant l(t_1) \\ \alpha_2(x_{i-l(t_1)}) & \text{if } l(t_1) < i \leqslant l(t_1) + l(t_2). \end{cases}$$

We define Λ_O to be the monoidal category having $(T_\tau(O), \otimes, (I, \varnothing))$ as its object structure such that there is a unique morphism $(t_1, \alpha) \to (t_2, \alpha)$ in Λ_O if and only if $l(t_1) = l(t_2)$. This unique morphism will be called the *O-association symbol* (a-symbol, for short) $(t_1, \alpha) \to (t_2, \alpha)$, which is denoted by $a(t_1, t_2, \alpha)$. Using the analogy of Proposition 3.1, it is easy to see that Λ_O is freely generated by the object variables (sorts) O.

Proposition 3.2. For every monoidal category $\mathscr{C}$ and any mapping ϕ from O to the objects of $\mathscr{C}$, there exists a unique strict monoidal functor $\Lambda_O \to \mathscr{C}$, also denoted by ϕ, sending (x_1, A) to $\phi(A)$ for each object variable A.

Now let $\mathscr{O} = (O, \otimes, I)$ be a τ-algebra, that is, a class O of *objects* with an interpretation of the τ-operations on O. Considering objects as object variables, we can still speak of object terms over O. For an object term $u = (t, \alpha)$ in $T_\tau(O)$, let $|u|_{\mathscr{O}}$ (or simply $|u|$, if $\mathscr{O}$ is understood) be the evaluation of u in $\mathscr{O}$ as an object in O. If $\rho = a(t_1, t_2, \alpha)$ is any O-association symbol $u_1 \to u_2$, then we also say that ρ is an $\mathscr{O}$-association symbol from $|u_1|$ to $|u_2|$, and, abusing the notation somewhat, $\rho : |u_1| \Rightarrow |u_2|$. Bear in mind that two $\mathscr{O}$-association symbols may not be composed as 'morphisms', as might be thought by this way of writing them. In general, it makes no sense to speak of the category $\Lambda_{\mathscr{O}}$ of $\mathscr{O}$-association symbols. Nevertheless, by Proposition 3.2, each a-symbol $\rho : A \Rightarrow B$ has a unique canonical interpretation as a real associator morphism $\rho : A \to B$ in every monoidal category $\mathscr{C}$ having $\mathscr{O}$ as its object structure.

Now consider $T_\tau(O)$ as the base collection of object variables, and concentrate on the monoidal category $\Lambda_{T_\tau(O)}$. The objects in this category are pairs $(t, \boldsymbol{\alpha})$, where t is a term in $\tilde{T}_\tau(X)$ and $\boldsymbol{\alpha} : X_n \to T_\tau(O)$ is a mapping. By Proposition 3.2, there exists a unique 'syntactical' strict monoidal functor $\Lambda_O : \Lambda_{T_\tau(O)} \to \Lambda_O$ extending the 'identity' injection $(x_1, u) \mapsto u$. Technically, $\Lambda_O a(t_1, t_2, \boldsymbol{\alpha})$ is obtained by substituting the sequence of trees $u_1, \ldots, u_n$ determined by $\boldsymbol{\alpha}$ into t_1 and t_2. Given the algebra $\mathscr{O}$, the result is viewed as an $\mathscr{O}$-association symbol $A \Rightarrow B$ for appropriate objects A and B. On the other hand, the a-symbol $a(t_1, t_2, \boldsymbol{\alpha})$ gives rise 'semantically' to the $\mathscr{O}$-association symbol

$$a(t_1, t_2, \boldsymbol{\alpha})/\mathscr{O} = a(t_1, t_2, \gamma) : A' \Rightarrow B',$$

where $\gamma = (|u_1|, \ldots, |u_n|)$. By the algebraic laws of tree substitution, $A = A'$ and $B = B'$. We say that the a-symbols

$$\rho_1 = \Lambda_O a(t_1, t_2, \boldsymbol{\alpha})$$
$$\rho_2 = a(t_1, t_2, \boldsymbol{\alpha})/\mathscr{O}$$

are *equivalent*, and write $\rho_1 \equiv \rho_2$. We say ρ_1 is a *refinement* of ρ_2. The point is that ρ_1 and ρ_2 define the same associator morphism $A \to B$ in every monoidal category having the object structure $\mathscr{O}$. In general, it is not easy to give a concrete characterisation of the equivalence $\equiv$ as the symmetric and transitive closure of refinement. Fortunately, however,

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

we shall not need any such characterisation in our coherence axiom I3 in Section 4.1 below, since simple refinement will suffice as $\equiv$ in that axiom.

3.2. *Permutation symbols*

Let O and $\mathscr{O} = (O, \otimes, I)$ be as in Section 3.1.

Definition 3.1. An *O-permutation symbol* is a quadruple (t_1, t_2, α, χ), where:
— $t_1, t_2 \in \tilde{T}_\tau(X)$ are of the same length n;
— $\alpha : X_n \to O$ is a mapping; and
— $\chi : n \to n$ is a permutation.

As a concrete permutation symbol (p-symbol, for short), the quadruple above will be identified as $p(t_1, t_2, \alpha, \chi)$. Extending the monoidal category Λ_O of O-association symbols, the symmetric monoidal category Π_O of O-permutation symbols is constructed as follows. Objects in Π_O are the same as in Λ_O, whereas morphisms $(t_1, \alpha) \to (t_2, \alpha')$ are p-symbols $p(t_1, t_2, \alpha, \chi)$ such that $\alpha(i) = \alpha'(\chi(i))$ for all $i \in [n] = \{1, \ldots, n\}$, where n is the common length of t_1 and t_2. Technically speaking, the object variables appearing on the leaves of the tree $u_1 = (t_1, \alpha)$ are relabelled according to χ, and the tree structure itself is replaced by t_2. Composition and tensor in Π_O are defined by:

$$p(t_1, t_2, \alpha, \chi) \circ p(t_2, t_3, \alpha', \eta) = p(t_1, t_3, \alpha, \chi \circ \eta)$$
$$p(t_1, t_2, \alpha, \chi) \otimes p(q_1, q_2, \beta, \eta) = p(t_1 \otimes q_1, t_2 \otimes q_2, \alpha \otimes \beta, \chi \otimes \eta).$$

Remember that $\alpha' = \chi^{-1} \circ \alpha$, and $\chi \otimes \eta$ is the obvious tensor of χ and η. The identity on object $u = (t, \alpha)$ is $1_u = p(t, t, \alpha, id_n)$, and the symmetry $c_{u,v}$ for $u = (t, \alpha)$ and $v = (q, \beta)$ is $p(t \otimes q, q \otimes t, \alpha \otimes \beta, \pi_{n,m})$, where $n = l(t)$, $m = l(q)$, and $\pi_{n,m}$ is the block transposition $n + m \to m + n$. Notice that the uniqueness of morphisms between two given objects no longer holds in Π_O. However, by Mac Lane's coherence theorem for symmetric monoidal categories (Mac Lane 1963), we still have the counterpart of Proposition 3.2.

Proposition 3.3. For every symmetric monoidal category $\mathscr{C}$ and any mapping ϕ from O to the objects of $\mathscr{C}$, there is a unique strict symmetric monoidal functor $\phi : \Pi_O \to \mathscr{C}$ sending (x_1, A) to $\phi(A)$ for each object variable A in O.

Given the algebra $\mathscr{O}$, we can again classify an O-permutation symbol

$$p(t_1, t_2, \alpha, \chi) : u_1 \to u_2$$

as an $\mathscr{O}$-one $|u_1| \Rightarrow |u_2|$ based on the evaluation of the trees u_1 and u_2. Permutation symbols $\rho_1 : A \Rightarrow B$ and $\rho_2 : B \Rightarrow C$ are said to be *composable* if they are composable as morphisms in the category Π_O. The concept of refinement and equivalence is also adopted in the obvious manner: for every $T_\tau(O)$-permutation symbol $\rho = p(t_1, t_2, \boldsymbol{\alpha}, \chi)$ with $\boldsymbol{\alpha} = (u_1, \ldots, u_n)$, the p-symbol $\Pi_O \rho$ is a refinement of $\rho/\mathscr{O} = p(t_1, t_2, \gamma, \chi)$, where $\gamma = (|u_1|, \ldots, |u_n|)$. Remember that Π_O is the syntactical strict symmetric monoidal functor $\Pi_{T_\tau(O)} \to \Pi_O$ extending the injection $(x_1, u) \mapsto u$. The meaning of $\rho_1 \equiv \rho_2$ is again that ρ_1 and ρ_2 define the same permutation in every symmetric monoidal category $\mathscr{C}$ having the object structure $\mathscr{O}$.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

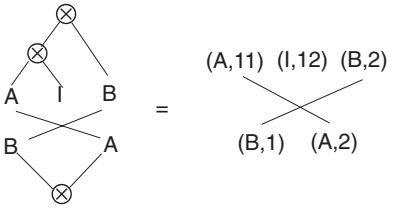


Fig. 3. Quasi-strict representation of p-symbols.

3.3. *Strict permutation symbols*

For strict monoidal categories, the form of permutation symbols becomes considerably simpler. Since the $\otimes$-tree structure need not be indicated, the objects of the category Π_O are simply finite sequences (strings) of object variables. Accordingly, a strict $\mathcal{O}$-permutation symbol is a pair (u, χ), where u is a string of object variables having length n and χ is a permutation $n \to n$. In the presence of a monoid $\mathcal{O}$ over O, the empty string as an object term is identified with I.

For notational convenience, we shall use a few instances of the symbols 1_u and $c_{u,v}$ in the non-strict setting as if they were strict, that is, as if the object terms u, v were strings and not trees. This can be done safely if the length of u and v is no more than 2 and I does not occur on the leaves of these trees. For example, 1_A, 1_{AB} and $c_{A,AB}$ will do as p-symbols

$$(x_1, A) \to (x_1, A),$$
$$(\otimes(x_1, x_2), (A, B)) \to (\otimes(x_1, x_2), (A, B))$$
$$(\otimes(x_1, \otimes(x_2, x_3)), (A, A, B)) \to (\otimes(\otimes(x_1, x_2), x_3), (A, B, A)),$$

respectively, but 1_{ABC} is already ambiguous as a non-strict identity. Taking this idea one step further, an object $u = (t, \alpha)$ of Π_O can be represented by a string of pairs

$$(A_1, p_1) \dots (A_n, p_n),$$

where each A_i is either an object variable or an occurrence of the symbol I (which is not supposed to be present in O). The second component p_i of (A_i, p_i) is the *path identifier* for leaf A_i in the tree u as a string of 1's and 2's – for example, $1 \dots 1$ identifies the leftmost leaf, and $2 \dots 2$ the rightmost one – see Figure 3. The set of all such paths in u will be denoted by $path(u)$, and $obj_u(p)$ will refer to the object variable (or I) sitting at leaf p of u. Path p is said to be *variable* if it points to an object variable, and *constant* if $obj_u(p) = I$. Then every p-symbol $p(t_1, t_2, \alpha, \chi) : u_1 \to u_2$ can be written in a unique way as a 'quasi'-strict p-symbol

$$(A_1, p_1) \dots (A_n, p_n) \to (B_1, q_1) \dots (B_m, q_m),$$

where p_i and q_j are appropriate paths in $path(u_1)$ and $path(u_2)$. With this way of writing the permutation, χ indeed appears as a permutation in the corresponding strict permutation symbol between the yields of u_1 and u_2 – see Figure 3 again. We shall need this representation of p-symbols in Section 5.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

4. Indexed monoidal algebras

In this section we introduce the category **IMA** of indexed monoidal algebras along the same lines as in Bartha and Jürgensen (1989), and establish an equivalence between the categories **IMA** and **SDCC**. From this point on in the paper, when we refer to a monoidal category, we will mean a symmetric one.

Going beyond the idea of indexing, the concept of indexed monoidal algebras is also very closely related to that of enriched categories (Kelly 1982). For an illustration we are going to recapture our SDCC category $(\mathbf{Rel}, \times)$ example as an enriched category. An *enriched category* has a *base* monoidal category $\mathscr{C}$ (in our case the category $(\mathbf{Rel}, \times)$ itself) from which the morphisms are adopted. Independently, there is a class R of objects, which in our example are sets. To each pair (A, B) of objects in R, we must assign a so-called *hom-object* $\mathscr{A}(A, B)$ of $\mathscr{C}$. We choose $\mathscr{A}(A, B) = A \times B$. Then we need to specify a *composition law*

$$M = M_{A,B,C} : \mathscr{A}(A, B) \otimes \mathscr{A}(B, C) \to \mathscr{A}(A, C)$$

as a collection of morphisms in $\mathscr{C}$. In our case, the choice is fairly straightforward in the form of a relation

$$(A \times B) \times (B \times C) \to (A \times C).$$

The *identity element* $j_A : I \to \mathscr{A}(A, A)$ is of course the identity relation on A. These data are subject to the conditions specified in Kelly (1982), which are trivially satisfied for our example. It follows that $\mathscr{C} = (\mathbf{Rel}, \times)$ can be recaptured as the *underlying category* of the $\mathscr{C}$-enriched category $\mathscr{A}$ defined in the above way. By definition, the underlying category has R as objects, and the morphisms $f : I \to \mathscr{A}(A, B)$ in $\mathscr{C}$ as morphisms $A \to B$. Observe that the idea is eventually to assign a collection of morphisms with each pair of objects (A, B), but this is done in an implicit way through the hom-functor $Hom(I, _)$ of the category $\mathscr{C}$. In our example this is exactly the covariant powerset functor, which is perfectly in line with the indexing idea presented in the previous section.

Of course, this way of recapturing an SDCC category $\mathscr{C}$ is trivial, and does not hide $\mathscr{C}$ itself at all. Therefore, we only adopt the idea of assigning a class of 'uni'-morphisms to each object (that is, the set $\mathscr{P}(A)$ to object A). The base category is dropped, and the composition law is replaced by the direct algebraic operations tensor and trace on morphisms. With the help of the indexing mechanism facilitated by the permutation symbols, these two operations will accomplish the same goal as the composition law in the base category – see also Bartha and Jürgensen (1989), where the composition law itself was captured by a general binary operation composition on morphisms.

4.1. *The definition of indexed monoidal algebras*

Formally, an *indexed monoidal algebra* (IMA, for short) consists of a class O of *objects* $A, B, C, \ldots$, a class M of *morphisms* $f, g, h, \ldots$, and an operation *rank*, which assigns to each morphism f an object A. We write $f : A$ to indicate the rank of f. There is also a distinguished unit object I and a binary operation $\otimes$ (tensor) on objects, which determine a τ-algebra $\mathscr{O} = (O, \otimes, I)$. As an early indication to the category structure we have in

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

mind, by $f : A \to B$, we mean a morphism $f : A \otimes B$ for each pair A, B of objects. The symbol $\otimes$ has become heavily overloaded by now. For this reason, we shall use $\oslash$ for $\otimes$ in permutation symbols. Accordingly, composition of p-symbols will be denoted by $\bullet$. Bear in mind, however, that, for example, the symmetry O-permutation symbol $c_{A,B} : \oslash(A, B) \to \oslash(B, A)$ is an $\mathcal{O}$-permutation symbol $A \otimes B \Rightarrow B \otimes A$ semantically.

With respect to M, the following operations are imposed:

— For each $\mathcal{O}$-permutation symbol $\rho : A \Rightarrow B$, a unary operation ρ, which assigns to each morphism $f : A$ a morphism $f \cdot \rho : B$.
— A binary operation tensor, which assigns to each pair of morphisms $f : A$ and $g : B$ a morphism $f \otimes g : A \otimes B$.
— A unary operation trace, which assigns to each morphism $f : (A \otimes A) \otimes B$ a morphism $\updownarrow_{A,B} f : B$.
— For each object A, a constant $\mathbf{1}_A : A \otimes A$.

We will write $\updownarrow_A f$ for $\updownarrow_{A,B} f$ whenever B is understood. Notice the boldface notation $\mathbf{1}_A : A \to A$ as opposed to $1_u : u \to u$ for permutation symbols.

We shall use two more 'categorical' operations, which are already derived from the above basic ones:

— A binary operation composition, which assigns to each pair of morphisms $f : A \to B$ and $g : B \to C$ the morphism

$$f \circ_{A,B,C} g = \updownarrow_B ((f \otimes g) \cdot (a_1 \bullet (c_{A,BB} \oslash 1_C) \bullet a_2) : A \to C,$$

where

$$a_1 : (A \oslash B) \oslash (B \oslash C) \to (A \oslash (B \oslash B)) \oslash C$$
$$a_2 : ((B \oslash B) \oslash A) \oslash C \to (B \oslash B) \oslash (A \oslash C)$$

are the obvious a-symbols. Observe that the prefix tree notation would actually be accurate for $\oslash$, but the infix one is easier to read in the present context – see Figure 4a.
— A binary operation (general) tensor, assigning to each pair of morphisms $f : A \to B$ and $g : C \to D$, the morphism

$$f \otimes_{A,B,C,D} g = (f \otimes g) \cdot (a_1 \bullet ((1_A \oslash c_{B,C}) \oslash 1_D) \bullet a_2) : (A \otimes C) \otimes (B \otimes D)$$

where

$$a_1 : (A \oslash B) \oslash (C \oslash D) \to (A \oslash (B \oslash C)) \oslash D$$
$$a_2 : (A \oslash (C \oslash B)) \oslash D \to (A \oslash C) \oslash (B \oslash D).$$

See Figure 4b.

Again, despite the ambiguity, we shall not indicate the indices in $\circ$ and $\otimes$ unless it is absolutely necessary. Clearly, the basic operation $\otimes$ is intended to be the instance $\otimes_{I,A,I,B}$ of the general operation. Observe that the above definition of composition and tensor is in line with the traced monoidal category axioms – see also Bartha (1987a, Identity $X3$) regarding composition. As we shall point out in Theorem 4.1, our trace operation models the canonical trace concept in SDCC categories.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press



(a) (b)

Fig. 4. Composition (a) and tensor (b) in $\mathscr{M}$.

The above operations are subject to the following equational axioms:

I1: *Functoriality of indexing:*

$$f \cdot (\rho_1 \bullet \rho_2) = (f \cdot \rho_1) \cdot \rho_2 \qquad \text{for } f : A \text{ and composable } \rho_1 : A \Rightarrow B,\ \rho_2 : B \Rightarrow C$$
$$f \cdot 1_A = f \qquad \text{for } f : A.$$

I2: *Naturality of indexing:*

$$(f \otimes g) \cdot (\rho_1 \oslash \rho_2) = f \cdot \rho_1 \otimes g \cdot \rho_2 \qquad \text{for } f : A,\ g : B,\ \rho_1 : A \Rightarrow C,\ \rho_2 : B \Rightarrow D$$
$$(\updownarrow_A f) \cdot \rho = \updownarrow_A (f \cdot (1_{AA} \oslash \rho)) \qquad \text{for } f : (A \otimes A) \otimes B,\ \rho : B \Rightarrow C.$$

I3: *Coherence:*

$$f \cdot \rho_1 = f \cdot \rho_2 \qquad \text{for } f : A, \text{ whenever } \rho_1 \equiv \rho_2.$$

I4: *Associativity and symmetry of tensor:*

$$((f \otimes g) \otimes h) \cdot a_{A,B,C} = f \otimes (g \otimes h) \qquad \text{for } f : A,\ g : B,\ h : C$$
$$f \otimes g = (g \otimes f) \cdot c_{B,A} \qquad \text{for } f : A,\ g : B.$$

I5: *Right identity (yanking):*

$$f \circ \mathbf{1}_B = f \qquad \text{for } f : A \to B.$$

I6: *Symmetry of identity:*

$$\mathbf{1}_A \cdot c_{A,A} = \mathbf{1}_A.$$

I7: *Vanishing:*

$$\updownarrow_I f = f \cdot a \qquad \text{for } f : (I \otimes I) \otimes A,$$

where $a : (I \oslash I) \oslash A \to A$

$$\updownarrow_{A \otimes B} (f \cdot a) = \updownarrow_B (\updownarrow_A f \cdot (1_A \oslash (c_{B,A} \oslash 1_{BC}))) \qquad \text{for } f : A \otimes ((B \otimes A) \otimes (B \otimes C)),$$

where $a : A \oslash ((B \oslash A) \oslash (B \oslash C)) \to ((A \oslash B) \oslash (A \oslash B)) \oslash C$.

I8: *Superposing:*

$$\updownarrow_A ((f \otimes g) \cdot a) = \updownarrow_A f \otimes g \qquad \text{for } f : (A \otimes A) \otimes B,\ g : C,$$

where $a : ((A \oslash A) \oslash B) \oslash C \to (A \oslash A) \oslash (B \oslash C)$.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

I9: *Trace swapping:*

$$\updownarrow_B (\updownarrow_A (f \cdot a)) = \updownarrow_A (\updownarrow_B (f \cdot ((c_{AA,BB} \oslash 1_C) \bullet a')) \quad \text{for} f : ((A \otimes A) \otimes (B \otimes B)) \otimes C,$$

$$\text{where } a : ((A \oslash A) \oslash (B \oslash B)) \oslash C \to (A \oslash A) \oslash ((B \oslash B) \oslash C)$$
$$a' : ((B \oslash B) \oslash (A \oslash A)) \oslash C \to (B \oslash B) \oslash ((A \oslash A) \oslash C).$$

The collection of axioms I1, …, I9 will be denoted by IM.

Let $\mathscr{M} = (O, M)$ and $\mathscr{M}' = (O', M')$ be indexed monoidal algebras. An *indexed monoidal homomorphism* $h : \mathscr{M} \to \mathscr{M}'$ maps each object A in O to an object hA in O' and each morphism $f : A$ in M to a morphism $hf : hA$ in M', so h defines a τ-algebra homomorphism $\mathscr{O} \to \mathscr{O}'$ between the object structures. Furthermore, the following homomorphism conditions are met by the morphisms:

(i) $h(f \cdot \rho) = (hf) \cdot h\rho$

(ii) $h(f \otimes g) = hf \otimes hg$

(iii) $h(\updownarrow_U f) = \updownarrow_{hU} hf$

(iv) $h\mathbf{1}_A = \mathbf{1}_{hA}$.

Notice that the symbol h has three different meanings in (i)–(iv) above: hA, hf and $h\rho$. In $h\rho$, h stands for the unique strict monoidal functor $\Pi_O \to \Pi_{O'}$ determined by h on objects. The category **IMA** consists of all indexed monoidal algebras as objects and indexed monoidal homomorphisms as morphisms.

We will now introduce our second motivating example, which is the single-sorted indexed monoidal algebra of open undirected graphs.

Example 4.1. An *open graph* (Bartha and Krész 2003) is an undirected multigraph (Lovász and Plummer 1986) with some of its vertices distinguished and labelled, each with a different label. In the present discussion we assume that the distinguished vertices have degree 1. We call these vertices, as well as the edges incident at them, *external*. If the number of external vertices in a concrete graph G is n, we use the numbers in $[n]$ to label them. Graph G is then of rank n, that is, $G : n$. Two open graphs are *isomorphic* if they are such as ordinary graphs by an isomorphism that preserves the labelling of the external vertices.

Clearly, the algebra $\mathscr{O}$ in this example is the monoid $(N, +, 0)$, which is isomorphic to $\{1\}^*$. Thus, the example is single-sorted and strict. The operations are interpreted on morphisms (open graphs) as follows:

— For a graph $G : n$ and permutation symbol $\rho : n \Rightarrow n$, the graph $G \cdot \rho$ is the graph obtained from G by relabelling its external vertices according to the ordinary permutation $n \to n$ determined by ρ in the strict single-sorted monoidal category of permutations. This permutation is the unique maximal refinement of ρ, which exists in the single-sorted setting.

— For graphs $G : n$ and $H : m$, we have that $G \otimes H : n + m$ is the disjoint union of G and H with the labels of H's external vertices incremented by n.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

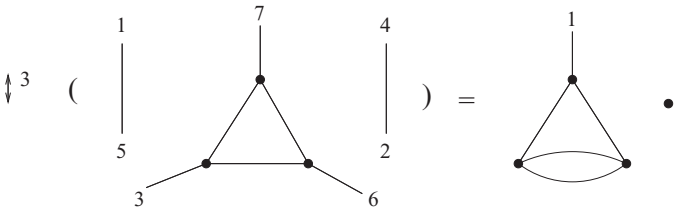


Fig. 5. Taking the trace of a graph.

— For every $n \in N$, the graph $\mathbf{1}_n : n + n$ consists of n edges connecting external vertex i with $n + i$ for each $i \in [n]$.
— For a graph $G : n + n + m$, we have that $\updownarrow_n G$ is constructed from G by gluing together the pairs of external edges ending in external vertices i and $n + i$ for each $i \in [n]$, and then discarding the vertices i and $n + i$ themselves. As part of this procedure, whenever a number of external edges are glued together in a cycle with no intercepting internal vertices, a new isolated vertex is added to the graph. Finally, the label of each remaining external vertex is decremented by $2n$. See Figure 5 for the implementation of $\updownarrow_3$ on a graph $G : 3 + 3 + 1$.

The algebra of open graphs defined in this way is denoted by $\mathscr{G}$. It is easy to verify that $\mathscr{G}$ is an IMA. The algebra of relations described in Section 2 is also indexed monoidal over sets as objects and pairs of sets (A, R) as morphisms such that $R \subseteq A$. Clearly, the rank of (A, R) is A.

4.2. *Equivalence of the categories* **IMA** *and* **SDCC**

The two examples discussed in Section 4.1 served as models for the definition of indexed monoidal algebras. The graph example reflects a 'coproduct' philosophy regarding the tensor $\otimes$, while in the relation example, the tensor is clearly product-oriented. However, the tensor will not become a coproduct or product in the corresponding SDCC category in either of these examples.

Theorem 4.1. The categories **IMA** and **SDCC** are equivalent.

Proof. Let $\mathscr{M} = (O, M)$ be an IMA, and define the monoidal category $\mathscr{C} = \mathscr{SM}$ over the objects O as follows. The morphisms $A \to B$ and identities in $\mathscr{C}$ are exactly those in $\mathscr{M}$, while composition and tensor are adopted from $\mathscr{M}$ as derived operations. For a morphism $f : C$ and objects A, B such that $C = A \otimes B$, we may want to use the distinction $f_{A,B} : A \to B$ to keep different hom-sets disjoint. We shall return to this foundational issue shortly. Symmetries $c_{A,B} : A \otimes B \to B \otimes A$ in $\mathscr{C}$ are the morphisms $\mathbf{1}_{A \otimes B} \cdot (1_{AB} \oslash c_{A,B})$. In general, every permutation symbol $\rho : u \to v$ is represented in $\mathscr{SM}$ as $\mathbf{1}_{|u|} \cdot (1_u \oslash \rho) : |u| \to |v|$. For each self-adjunction $A \dashv A$, the unit map $d_A : I \to A \otimes A$ and the counit map $e_A : A \otimes A \to I$ are $\mathbf{1}_A \cdot l^{-1}$ and $\mathbf{1}_A \cdot r^{-1}$, respectively, where l and r are the appropriate left and right unitors.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

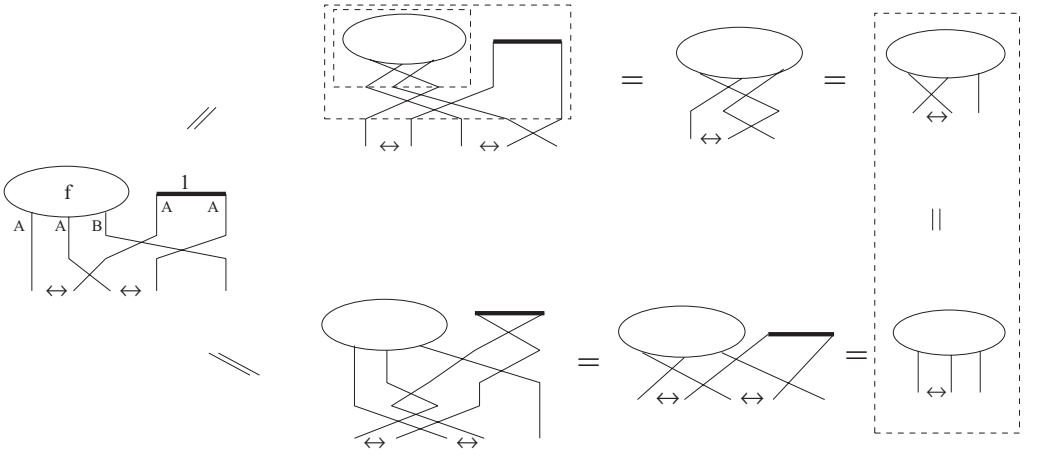


Fig. 6. Symmetry of trace.

Essentially, it is routine to check that $\mathscr{SM}$ is an SDCC category, though some of the details require careful organisation. We start out with a few immediate consequences of the axioms IM.

J1: *Symmetry of trace*

$$\updownarrow_A f = \updownarrow_A (f \cdot (c_{A,A} \oslash 1_B)) \quad \text{for } f : (A \otimes A) \otimes B.$$

See Figure 6.

J2: *Canonical trace*

$$\updownarrow_A f = ((\mathbf{1}_A \cdot l^{-1}) \circ_{I,(A\otimes A),B} f) \cdot l' \quad \text{for } f : (A \otimes A) \otimes B$$

where l and l' are appropriate left unitors.

See Figure 7.

Notice that in the diagrams of Figures 6 and 7 we rely heavily on axiom I3 (coherence), taking refinements of permutation symbols whenever this becomes necessary. For technical simplicity we work in the strict monoidal setting in these diagrams. Filling in the parentheses and the corresponding association symbols to make these diagrams work in the non-strict case is left as an easy exercise.

J3: *Left identity*

$$\mathbf{1}_A \circ f = f \quad \text{for } f : A \to B.$$

See Figure 8. Note that the symmetry of trace and that of $\mathbf{1}_A$ have both been used in the proof.

J4: *Tensor of identity*

$$\mathbf{1}_{A\otimes B} = \mathbf{1}_A \otimes_{A,A,B,B} \mathbf{1}_B.$$

See Figure 9, and take $f = \mathbf{1}_{A\otimes B}$.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

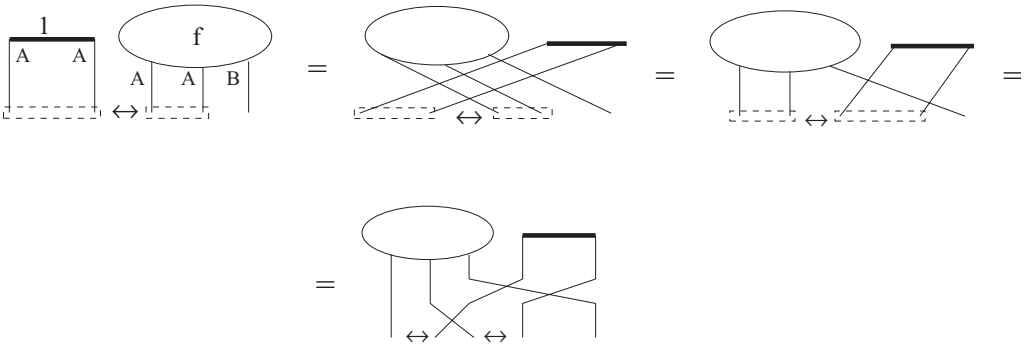


Fig. 7. Canonical trace (continue on Figure 6 bottom).

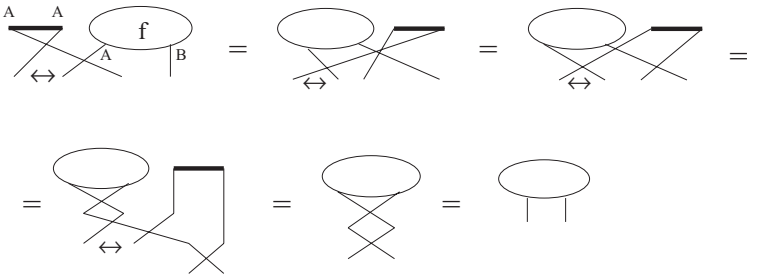


Fig. 8. Left identity.



Fig. 9. Tensor of identity.

With regard to the category $\mathscr{C}$, the monoidal axioms follow trivially from I1, …, I6, J3 and J4. Proving the unit and counit property of d_A and e_A is equally easy. See, for example, Figure 10 for the first equation in Definition 2.2. The guiding principle to follow in understanding this diagram is to note that whenever either endpoint of an instance of $\mathbf{1}_A$ is involved in the trace operation $\updownarrow_A$, that instance can be eliminated from the diagram by 'yanking'. This rule is a direct consequence of axioms I5, I6 and J3 in light of the indexing mechanism. The definition of functor $\mathscr{S}$ on (homo-)morphisms is evident, and left as an exercise.

Conversely, let $\mathscr{C}$ be an SDCC category having the object structure $\mathscr{O}$. Define the IMA $\mathscr{M} = \mathscr{I}\mathscr{C}$ as follows. For each object A, the morphisms of rank A are those in $\mathscr{C}(I, A)$. Since $\mathscr{C}$ is symmetric, every permutation symbol $\rho : A \Rightarrow B$ determines a permutation

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

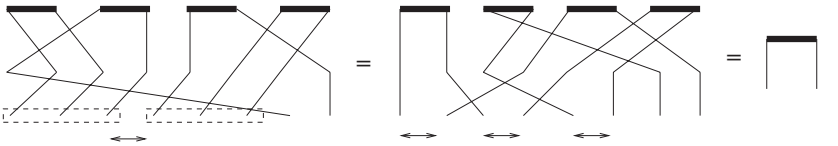


Fig. 10. Proving $\mathscr{S}\mathscr{M}$ compact closed.

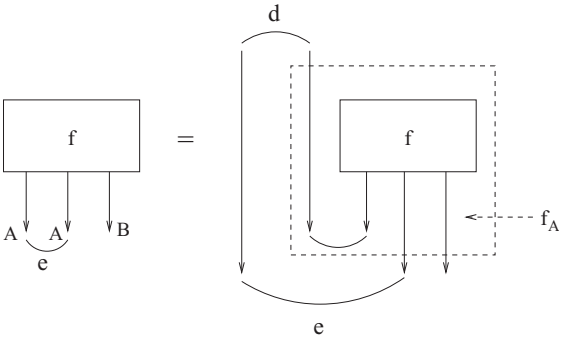


Fig. 11. Canonical trace again, cf. Figure 2.

$\rho_{\mathscr{C}} : A \to B$ in $\mathscr{C}$. Then, for $f : A$, define $f \cdot \rho = f \circ_{\mathscr{C}} \rho_{\mathscr{C}}$. Observe that indexing indeed becomes the restriction of the covariant hom-functor to permutations, as intended. For $f : A$ and $g : B$ in $\mathscr{M}$, let

$$f \otimes_{\mathscr{M}} g = l^{-1} \circ (f \otimes_{\mathscr{C}} g) : I \to A \otimes B,$$

where l is the unitor $I \otimes I \to I$. As for the identities, we let $\mathbf{1}_A = d_A : I \to A \otimes A$. For $f : (A \otimes A) \otimes B$, we define $\updownarrow_A f$ as the canonical trace of the morphism $f_A : A \otimes I \to A \otimes B$ in $\mathscr{C}$ that corresponds to $f \circ a_{A,A,B}$ according to compact closure. That is, $\updownarrow_A f$ is the morphism

$$f \circ (e_A \otimes 1_B) = (d_A \otimes 1_I) \circ a_1 \circ (1_A \otimes f_A) \circ a_2 \circ (e_A \otimes 1_B) : I \to B$$

in $\mathscr{C}$ with the associators a_1 and a_2 as appropriate – see Figure 11.

In view of this translation, each of the equations in IM is either a standard monoidal category axiom or has already been observed in Joyal *et al.* (1996) and Kelly and Laplaza (1980) for traced monoidal or compact closed categories. Thus, $\mathscr{M}$ is an IMA. The definition of functor $\mathscr{I}$ on morphisms (strict monoidal functors) is again straightforward.

By definition, $\mathscr{I}(\mathscr{S}\mathscr{M}) \cong \mathscr{M}$ through indexing by an appropriate unitor. On the other hand, the only difference between the SDCC categories $\mathscr{C}$ and $\mathscr{S}(\mathscr{I}\mathscr{C})$ is that the hom-sets $A \to B$ in the latter are identified with the hom-sets $I \to A \otimes B$ of the former, using the natural isomorphisms given by the self-adjunctions $A \dashv A$. In other words, morphisms $A \to B$ in $\mathscr{S}(\mathscr{I}\mathscr{C})$ – as provided for by compact closure – are simply renamed as they appear in $\mathscr{C}(I, A \otimes B)$. Thus, there exists a natural isomorphism between the functors $1_{\mathbf{SDCC}}$ and $\mathscr{S}\mathscr{I}$, so the categories **IMA** and **SDCC** are indeed equivalent as stated. □

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

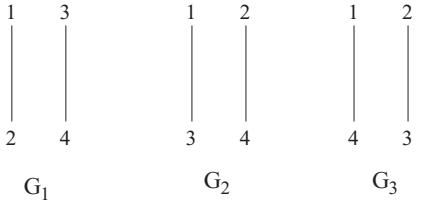


Fig. 12. Three simple graphs.

Example 4.2. There are exactly three non-isomorphic open graphs G_1, G_2 and G_3 of rank 4 in $\mathscr{G}$ that consist of external vertices only. These graphs are given in Figure 12. Clearly, $G_1 = \mathbf{1}_1 \otimes \mathbf{1}_1$ and $G_2 = \mathbf{1}_1 \otimes_{1,1,1,1} \mathbf{1}_1$. Graph G_3 is most simply expressed as $c_{1,1} : 2 \to 2$ in the SDCC category $\mathscr{SG}$. Following along these lines, it is instructive to find simple terms for each graph as morphisms $i \to j$ $(i + j = 4)$ in $\mathscr{SG}$. For example,

$$G_1 = e_1 \otimes d_1 : 2 \to 2$$
$$G_2 = (1_1 \otimes d_1) \circ (c_{1,1} \otimes 1_1) : 1 \to 3$$
$$G_3 = 1_1 \otimes e_1 : 3 \to 1.$$

Example 4.2 illustrates the difficulty in interpreting SDCC-terms as open graphs, and *vice versa*. The graphical language (Joyal and Street 1991; Selinger 2009) for CC categories clarifies this translation procedure to some extent, but the root of the problem is that the monoidal category language assumes (explicitly or implicitly) that hom-sets are pairwise disjoint. According to Mac Lane (Mac Lane 1971), however, this is not a necessary requirement if one uses the hom-set-based definition of categories. The domain and codomain of a concrete morphism f are only fixed by the standard definition of categories. Consider, for example, the CC category $Int(\mathscr{C})$ freely generated by a traced monoidal category $\mathscr{C}$ as described in Joyal *et al.* (1996). This category has as objects pairs (A, B) of objects in $\mathscr{C}$. Morphisms $(A, B) \to (C, D)$ in $Int(\mathscr{C})$ are the morphisms $A \otimes D \to C \otimes B$ in $\mathscr{C}$. Clearly, hom-sets will overlap in $Int(\mathscr{C})$ unless we separate them by force. The very same idea manifests itself in indexed monoidal algebras, reflecting the philosophy that morphisms are fixed, while their domain and codomain may vary, depending on what hom-sets we want to put them in. It is only the rank of a morphism that is uniquely determined. This is the major difference between our graphical language, which is universal for IMA's, and the one described in Joyal and Street (1991) and Selinger (2009) for CC categories as symmetric autonomous categories.

A further remark on traced monoidal categories is in order at this point, and it will also explain the Int construction, too, through an intuitive argument. Consider the traced monoidal category $\vec{\mathscr{G}}$ of open *directed* graphs (flowcharts) as described in Bartha (1987a). In this category, a morphism $n \to m$ is an open graph G with $n + m$ external vertices such that each edge is directed in G. Accordingly, the external vertices are labelled by either b_i, $i \in [n]$ (begin vertices) if their out-degree is 1, or by ex_j, $j \in [m]$ (exit vertices) if their in-degree is one. As proved in Bartha (1987a), $\vec{\mathscr{G}}$ is freely generated by the directed star graphs (boxes) $n \to m$ as a traced monoidal category. Since $\mathscr{SG}$ is also traced with the

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

canonical trace, the mapping that takes each directed star graph $\vec{K}_{n,m} : n \to m$ to the undirected star graph $K_{n+m} : n + m$ (as a morphism $n \to m$ in $\mathcal{SG}$) can be extended in a unique way to a traced monoidal functor U. Clearly, $G_2 = U1_2$ and $G_3 = Uc_{1,1}$. There is, however, no morphism in $\vec{\mathcal{G}}$ that is mapped to G_1 by U. It is therefore misleading to say, at least in the categorical context, that $\mathcal{G}$ is obtained from $\vec{\mathcal{G}}$ by 'forgetting' the direction of the edges in graphs.

We will now revise the above argument in the context of the Int construction. For each star graph $\vec{K}_{n,m} : n \to m$, we divide the n in-degrees of the centre vertex into n_1 'real' in-degrees and m_2 'dual' out-degrees, so $n_1 + m_2 = n$. Symmetrically, we divide the m out-degrees into m_1 real out-degrees and n_2 dual in-degrees. Correspondingly, we distinguish between real and dual begin/exit vertices as appropriate, and say that $\vec{K}_{n,m}$ is a morphism $(n_1, n_2) \to (m_1, m_2)$ in the new setting. In general, a morphism $(n_1, n_2) \to (m_1, m_2)$ in the resulting category $Int(\vec{\mathcal{G}})$ is just a morphism $n_1 + m_2 \to m_1 + n_2$ in $\vec{\mathcal{G}}$. When composing graphs in $Int(\vec{\mathcal{G}})$, we will always connect real out-degrees with real in-degrees, and dual in-degrees with dual out-degrees simultaneously. Observe that this philosophy requires composition in $\vec{\mathcal{G}}$ on the real side and trace (feedback) on the dual side. See Joyal *et al.* (1996) for the details.

It follows from the general result in Joyal *et al.* (1996) that $Int(\vec{\mathcal{G}})$ is a CC category in which the dual of each object (n, m) is (m, n). Moreover, $Int(\vec{\mathcal{G}})$ is freely generated by $\vec{\mathcal{G}}$. Regarding our SDCC category $\mathcal{SG}$, it can be recaptured from $Int(\vec{\mathcal{G}})$ by taking the sub- (symmetric) monoidal category generated by the star graphs $\vec{K}_{k,k}$ $(k \geqslant 1)$ considered as morphisms $(n, n) \to (m, m)$ with $k = n + m$. Clearly, this subcategory stays within the scope of self-dual objects (n, n), $n \in N$, and it is isomorphic to $\mathcal{SG}$. Heuristically speaking, the Int construction suggests that $\mathcal{G}$ is obtained from $\vec{\mathcal{G}}$ by making the edges in directed graphs *bidirectional*. This seemingly trivial observation has far-reaching consequences in computer science with respect to the reversibility of computations. For example, the bus connection between the processor and the memory in a von Neumann computer is bidirectional, and the interconnections between tape cells in a Turing machine are bidirectional. We shall elaborate further on this point in Sections 6 and 7.

5. Coherence in indexed monoidal algebras

In general, a coherence result for some type μ of monoidal categories is concerned with establishing a left adjoint for a forgetful functor F from the category of μ-monoidal categories into an appropriate syntactical category, and providing a graphical characterisation of the free monoidal μ-categories so obtained – see Mac Lane (1971), Kelly and Laplaza (1980), Selinger (2009), Bartha (1987a) and Bartha (1987b) for some typical examples. We also presented two such results in Section 3 by constructing the monoidal categories Λ_O and Π_O. In this section we present a more substantial coherence theorem for SDCC categories, but phrase it in terms of indexed monoidal algebras. The graphical language arising from this result will justify our efforts in the previous section to reconsider SDCC categories in the given algebraic context.

Our way of choosing the forgetful functor F differs from the method followed in Kelly and Laplaza (1980), where the category structure was still preserved. We go one step

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

further in the forgetting, and preserve only the alphabet structure of morphisms. For a class O of object variables, a *ranked alphabet* (signature) $\Sigma = (O, M)$ consists of a class M of *morphism variables* and a mapping $M \to T_\tau(O)$ called *rank*. We again write $f : u$ to indicate the rank of f. Recall that $T_\tau(O)$ is the class of object terms over O. An *alphabet mapping* between ranked alphabets $\Sigma = (O, M)$ and $\Delta = (O', M')$ is a mapping ϕ that assigns to each object variable A in O an object variable ϕA and to each morphism variable $f : u$ in M a morphism variable $\phi f : \phi u$ in M'. Remember that ϕ also stands for the unique strict monoidal functor $\Pi_O \to \Pi_{O'}$ determined by ϕ. We say that an alphabet mapping *preserves the rank* of morphism variables.

Observe that our notion of a ranked alphabet is completely in line with the concept of monoidal signatures found in Joyal and Street (1991). By definition, a *monoidal signature* consists of a set Σ_0 of *object variables*, a set Σ_1 of *morphism variables* and a pair of functions *dom*, *cod* : $\Sigma_1 \to Mon(\Sigma_0)$, where $Mon(\Sigma_0)$ is the set of *object terms* over Σ_0. Object terms in our sense are exactly the same as those in Σ_0, and our morphism variables are 'one half' of the morphism variables in Σ_1.

Every IMA $\mathscr{M} = (O, M)$ can be considered trivially as a ranked alphabet $\Sigma = \mathscr{A}\mathscr{M}$ in which the object variables are O and the morphism variables with rank u are simply the morphisms in M of rank $|u|$. We use a subscript to distinguish between instances of morphism f belonging to different ranks as morphism variables. If $h : \mathscr{M} \to \mathscr{M}'$ is a homomorphism, then $\mathscr{A}h : \mathscr{A}\mathscr{M} \to \mathscr{A}\mathscr{M}'$ is the alphabet mapping ϕ by which $\phi A = hA$ and $\phi f_u = (hf)_{hu}$ for every morphism $f : |u|$ in $\mathscr{M}$. Our aim is to provide a left adjoint for the functor $\mathscr{A}$. In algebraic terms this amounts to constructing the IMA freely generated by Σ.

Let $\Sigma = (O, M)$ be a ranked alphabet, which will be fixed for the rest of this section. First we augment Σ by the following morphism variables:

— For each object term u, a symbol $H_u : u$.
— For each object variable A, a symbol $\perp_A : I$.

These symbols must not originally be present in Σ. Let Σ_H denote the augmented alphabet. The IMA $\mathscr{G}(\Sigma)$ freely generated by Σ has as objects $T_\tau(O)$ (that is, object terms over O), and as morphisms Σ-graphs, which are defined below. The τ-algebraic structure on objects is the free one.

We define a Σ-*graph* to be a finite undirected and labelled multigraph $G = (V, E, \ell)$ with vertices V, edges E and vertex labelling ℓ, such that for each vertex v, the label $\ell(v)$ of v is a morphism variable in Σ_H. Exactly one vertex, called the *host*, is labelled by the symbol H_u for some object term u, and this term identifies the rank of G as a morphism in the prospective IMA $\mathscr{G}(\Sigma)$.

Intuitively, G is a network of satellite machines of some sort that are interconnected with each other and the host as indicated by the edges in E. It is required that the label of each vertex v be consistent with its degree $d(v)$, so $d(v) = l(\ell(v))$. If $\ell(v) = u$, then v has a so-called *port* associated with each path $p \in path(u)$ (recall the definition of paths from Section 3.3). Port p is variable/constant if p is variable/constant as a path. Each variable port of vertex v identifies the point at which a unique edge impinges on v. To be meticulously precise about the term 'impinges on', v is in fact a coherent

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

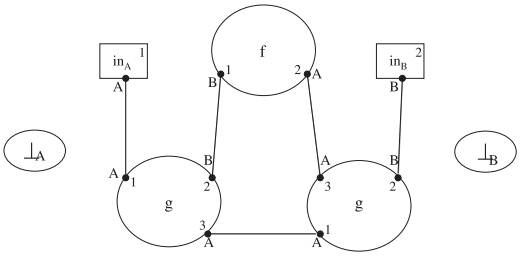


Fig. 13. A Σ-graph.

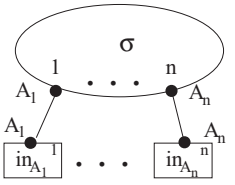


Fig. 14. An atomic Σ-graph

group of port vertices, each of which is labelled by a port identifier (that is, path p). In addition, the whole group of ports is labelled by $\ell(v)$. Edges, too, must be consistent with the labelling in the sense that for each edge connecting two different ports p_1 and p_2 belonging to vertices labelled by morphism variables of rank u_1 and u_2, respectively, we have $obj_{u_1}(p_1) = obj_{u_2}(p_2)$.

See Figure 13 for an example Σ-graph $G : AB$ in the strict setting, where the morphism variables f and g have ranks BA and ABA, respectively. The host itself is invisible in the figure, only its ports are indicated as separate boxes, which we call channels or *interfaces*. All other vertices, as well as the edges connecting them, are said to be *internal*. Vertices labelled $\perp_A$ are the *loop* vertices in G. Note that the idea of using a host vertex originates from Leiserson (1983).

Morphism variables in Σ are represented as *atomic* Σ-graphs (star graphs) in the way shown in Figure 14. An *isomorphism* between Σ-graphs $G, G' : u$ is a graph isomorphism that preserves the labelling information of the vertices. We shall not distinguish between isomorphic graphs. The indexed monoidal algebra operations are defined on Σ-graphs as follows:

— For a graph $G : u$, each $T_\tau(O)$-permutation symbol $\rho : u \Rightarrow u'$ is interpreted as the relabelling of the interfaces (that is, ports of the host) according to the O-permutation symbol $\Pi_O \rho$. Technically, this amounts to changing the label of the host together with the paths associated with the variable ports of the host, and the possible deletion/introduction of constant ports within that vertex. See Figure 3 again.

— For graphs $G_1 : u_1$ and $G_2 : u_2$, we get $G_1 \otimes G_2$ by taking the disjoint union of the subgraphs of G_1 and G_2 induced by their internal vertices, joining the two hosts into one labelled by $H_{u_1 \oslash u_2}$, adjusting the port labels of the new host in the obvious way

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

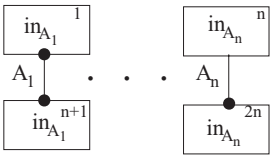


Fig. 15. The identity graph

and then connecting these ports to the respective ports in the internal copies of G_1 and G_2 in the same way as these edges exist in G_1 and G_2 separately.

— The identity graph $\mathbf{1}_u : u \oslash u$ is shown in Figure 15 in the strict setting, where u is the string $A_1 \dots A_n$. For a non-strict example, the graph $\mathbf{1}_I$ has the host as its sole vertex, which has two constant ports. In general, port $1w$ is connected to port $2w$ within the host whenever these ports are variable.

— For a graph $G : (u \oslash u) \oslash v$ with $l(u) = n$, the trace operation $\updownarrow_u$ is defined by gluing together each edge ending at interface labelled $11w$ with the edge ending at interface labelled $12w$ for all existing variable paths w in the tree u after detaching these edges from the host. (The first 1 in $11w$ and $12w$ points to the left at the root of the tree $\oslash(\oslash(u, u), v)$).) Whenever this procedure results in a loop of an even number of edges (but no intercepting internal vertices) glued together, a new loop vertex labelled by $\perp_A$ is created and added to the graph, where A is the common object variable associated with the interfaces involved in the loop (that is, $A = obj_{(u \oslash u) \oslash v}(p_i)$, where p_i is any of the ports/paths involved). Finally, the label of the host is changed to H_v and the port labels are adjusted accordingly.

Again, see Figure 5 and Bartha and Jürgensen (1989) for single-sorted examples. See also Elgot (1975), Bloom and Ésik (1985), Bartha (1987a) and Bartha (1987b) for the corresponding standard definition of feedback/iteration in (directed) flowcharts. Interestingly, in all of these references, graphs (flowcharts) are equipped with a single loop vertex, so loops do not multiply when taking the feedback. On the other hand, the loop vertex is present in the graph $\mathbf{1}_I$ as well. For the single-sorted case this amounts to imposing the additional axiom $\updownarrow \mathbf{1}_1 = \mathbf{1}_0$ (more precisely, its directed version, for example, Bartha (1987a, Axiom S5: $\uparrow 1 = 0$)), which is not a standard traced monoidal category axiom. However, from the point of view of axiomatisation, this is a minor issue. On the other hand, another issue, namely the assignment of an *individual monoid* to each object A, is extremely important and interesting. In terms of flowcharts, this allows us to erase begin vertices and join two incoming edges at any given port – see, for example, the constants $0_1 : 0 \to 1$ and $\epsilon : 2 \to 1$ in Bartha (1987a) and Bartha (1987b). In Joyal and Street (1991), these morphisms are called the *co-unit* and the *diagonal*, respectively; MacLane calls them η and μ (Mac Lane 1971); and according to Selinger (2009), they are the erase and copy maps. These constants (morphisms) were naturally incorporated in the axiomatisation of schemes – both flowchart and synchronous. For undirected graphs, the presence of such morphisms with a 'circularly symmetric' interface (for example, a fan into three equivalent directions for the diagonal $2 \to 1$) allows an upgrade of ordinary edges to hyperedges in exactly the same way as described in Milner (2009) for bigraphs. The

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press



Fig. 16. Two diagrams for 1_{A^*}.

axiomatisation of undirected hypergraphs as indexed monoidal algebras will be presented in a forthcoming paper.

It is easy to check that the above interpretation of the indexed monoidal operations on $\mathscr{G}(\Sigma)$ satisfies the axioms IM. Technically, $\mathscr{G}(\Sigma)$ is the multi-sorted, non-strict and labelled counterpart of the trivial IMA $\mathscr{G}$. Thus, $\mathscr{G}(\Sigma)$ is an IMA. It is also clear that $\mathscr{G}(\Sigma)$ is generated by Σ, that is, by the collection of the atomic Σ-graphs – see Figure 14 again. Indeed, every undirected graph can be reconstructed from its vertices as star graphs by adding internal edges one by one using the trace operation. Our goal is to show that $\mathscr{G}(\Sigma)$ is freely generated by Σ. In the proof, we are going to follow the standard algebraic technique of working in the totally free algebra of well-formed Σ-terms in order to take the quotient of this algebra determined by the given set IM of identities. Observe that the axioms IM are indeed identities, since each permutation symbol is an individual unary operation in our framework.

Compared with the simple-looking graphical language for CC categories described in Selinger (2009), our concept of a Σ-graph may appear to be unduly meticulous, but we believe it is not. The distinction between ports and their labelling is necessary in order to indicate clearly the flow of information in the graph, even in the strict case. Yet a graph must remain a graph in the standard combinatorial sense. Consider, for example, the two diagrams in Figure 16, which both represent the identity morphism 1_{A^*} in Selinger's graphical language. Note that Selinger labels the edges by object variables, rather than distinguishing ports and labelling those. The graphical language itself originates from Joyal and Street (1991), where the geometric aspects of such graphs are better explained. Selinger's coherence statement (Selinger 2009, Theorem 4.33) for CC categories says:

> 'A well-formed equation between morphisms in the language of CC categories follows from the axioms of CC categories if and only if it holds, up to isomorphism of diagrams, in the graphical language.'

The isomorphism of the diagrams of Figure 16 as 'graphs' is far from obvious. This leads to an ambiguity in the understanding of the coherence statement with respect to the direction and labelling of the arrows in diagrams, which is not easy to clarify. For this reason we do not rely on Selinger's graphical language for CC categories in our coherence statement. Another reason is that our concern is with indexed monoidal algebras and not with SDCC categories in the first instance.

The structure $T(\Sigma) = (T(\Sigma)_u | u \in T_\tau(O))$ of *well-formed indexed monoidal* Σ*-terms* (Σ-terms, for short) is the algebra (not IMA) freely generated from the morphism variables in Σ using the indexed monoidal operations and constants. Thus, $T(\Sigma)$ is a totally syntactical many-sorted term algebra in which the operations remain uninterpreted. Interpreting a Σ-term t in the algebra $\mathscr{G}(\Sigma)$ results in a Σ-graph $|t|$. Our goal is to show that an equation $t = t'$ in $T(\Sigma)$ is provable from the axioms IM if and only if $|t| = |t'|$. The proof of this statement will revisit the normal form construction in Bartha (1987a) by adjusting

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press



Fig. 17. Normal form of Σ-terms.

it to the present undirected environment in a straightforward manner. Part of this work has already been done in Bartha and Jürgensen (1989). Without any essential loss of generality, we will restrict the proof to the strict single-sorted case. Working under this assumption, the subscript 1 in $\mathbf{1}_1$, $\updownarrow_1$ and $\perp_1 = \updownarrow_1 1_1$ will be understood and omitted. The notation $\updownarrow^l$ stands for the l-fold application of $\updownarrow$. The alphabet Σ collapses into an ordinary ranked alphabet $(\Sigma_n \mid n \in N)$.

Definition 5.1 (Bartha 1987a, Definition 5). A Σ-term $t : n$ (that is, $t \in T(\Sigma)_n$) is in *normal form* (n.f., for short) if

$$t = \updownarrow^l (((\otimes_{i=1}^{p} \sigma_i) \otimes (\otimes_{j=1}^{q} \mathbf{1})) \cdot \rho) \otimes (\otimes_{k=1}^{r} \perp)$$

for some $l, p, q, r \in N$, $\sigma_i \in \Sigma_{m_i}$ and permutation (symbol) ρ such that:

(1) $n = m + 2q - 2l$, where $m = \Sigma_i m_i$, and
(2) $\rho : m + 2q \to m + 2q$ is *l-straight* in the sense that:

(a) For each $j \in [q]$,

$$2l < \rho(m + 2j - 1) < \rho(m + 2j).$$

Furthermore,

$$\rho(m + 2j - 1) < \rho(m + 2j + 1),$$

provided $j < q$.

(b) For each $i \in [l]$,

$$\rho^{-1}(2i - 1) < \rho^{-1}(2i).$$

Furthermore,

$$\rho^{-1}(2i - 1) < \rho^{-1}(2i + 1),$$

provided $i < l$.

See Figure 17 for the case where $l = p = q = r = 2$, $\sigma_1 : 3$ and $\sigma_2 : 2$.

The intuitive meaning of Definition 5.1 should be clear. The only 'ugly' technical issue is the l-straight property of ρ, which is intended to resolve the ambiguity caused by the symmetry of $\mathbf{1}$ (condition 2(a)) and that of trace (condition 2(b)). Obviously, no occurrence of $\mathbf{1}$ is allowed to be involved in the trace operation, which condition is also ensured by the first inequality of 2(a).

Now we prove the undirected counterpart of Bartha (1987a, Lemma 6).

Lemma 5.1. For every Σ-term $t : n$, there exists a Σ-term t' in n.f. such that $t = t'$ is provable from IM.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

Proof. First we show that t can be transformed into a term t'' of the form

$$t'' = \updownarrow^l ((\otimes_{i=1}^m a_i) \cdot \rho),$$

where for each $i \in [m]$, $a_i \in \Sigma$ or $a_i = \mathbf{1}$. Such a term is said to be in *weak normal form.* The proof of this statement uses a simple induction on the structure of t. The details of this argument can be found in Bartha (1987a, Lemma 5). Obtaining an n.f. t' from a weak n.f. t'' then reduces to an easy exercise. One must rearrange the terms a_i in a proper order using the symmetry of $\otimes$, eliminate occurrences of $\mathbf{1}$ that are involved in the trace operation by yanking, and adjust ρ to be l-straight by relying on the symmetry of $\mathbf{1}$ and that of trace. □

Theorem 5.2 (Bartha 1987a, Theorem 1). Let t and t' be Σ-terms such that $|t| = |t'|$. Then the equation $t = t'$ is provable from IM.

Proof. By Lemma 5.1, we can assume that t and t' are in n.f. The normal form of Σ-terms was defined in such a way that the only difference between t and t' may appear in the order of the atomic Σ-terms occurring in $\otimes_{i=1}^m \sigma_i$. Discrepancies of this nature can, however, be eliminated by the symmetry of $\otimes$ and the indexing mechanism. □

Corollary 5.3. The algebra $\mathscr{G}(\Sigma)$ is freely generated by Σ.

Proof. The statement is immediate from Theorem 5.2. □

We will now return to the discussion following Example 4.2 to extend the remark made there in connection with the Int construction. On the basis of that discussion, we can characterise the algebra $\mathscr{G}(\Sigma)$ as the IMA corresponding to an appropriate subcategory $\mathscr{C}$ of the CC category $Int(Sch(\bar{\Sigma}))$, where $Sch(\bar{\Sigma})$ is the traced monoidal category of $\bar{\Sigma}$-flowchart schemes described in Bartha (1987a) over the doubly ranked alphabet $\bar{\Sigma}$ in which $\bar{\sigma} \in \bar{\Sigma}_{n,n}$ if and only if $\sigma \in \Sigma_n$. Unfortunately, we cannot use this characterisation directly to prove Corollary 5.3. Indeed, let $\mathscr{M}$ be an arbitrary (single-sorted) IMA. Then, due to the universality of the Int construction and that of $Sch(\bar{\Sigma})$ as a traced monoidal category, there is a unique strict monoidal functor from the CC category $Int(Sch(\bar{\Sigma}))$ to $\mathscr{SM}$ extending any given rank-preserving mapping of $\bar{\Sigma}$ into $\mathscr{SM}$. The problem is, however, that the ports of the atomic Σ-graphs have been duplicated, so the image of a morphism (bidirectional flowchart) $\vec{G}$ from the subcategory $\mathscr{C}$ will also be a $\otimes$-duplicated version of the morphism f_G in $\mathscr{SM}$ that we intend to produce as the image of the corresponding undirected graph G. Since there is no way to retrieve f_G from $f_G \otimes f_G$, the method will eventually fail. This fallacy indicates that the construction of the SDCC category freely generated by an arbitrary traced monoidal category is not a simple matter, even with the Int construction in hand.

Definition 5.2. Let $\mathscr{M}$ be an arbitrary indexed monoidal algebra. An *interpretation* of Σ in $\mathscr{M}$ is an alphabet mapping $\Omega : \Sigma \to \mathscr{AM}$.

By Corollary 5.3, every interpretation Ω can be extended in a unique way to a homomorphism $\Omega : \mathscr{G}(\Sigma) \to \mathscr{M}$. Thus, $\mathscr{G}$ is indeed a left adjoint for the functor $\mathscr{A}$.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

6. Turing automata and Turing graph machines

In this section we introduce the algebra of Turing automata and Turing graph machines as an important example of indexed monoidal algebras. We shall use the monoidal category (**Bset**, $+$) (sets and bijections with disjoint union as tensor) as the hypothetical index category. For a set A, let $A_\star = A + \{\star\}$, where $\star$ is a fixed symbol called the *anchor*.

Definition 6.1. A *Turing automaton* (TA, for short) $T : A$ is a triple (A, Q, δ), where A is a set of *interfaces*, Q is a non-empty set of *states* and $\delta \subseteq (Q \times A_\star)^2$ is the *transition relation*.

Two automata (A, Q, δ) and (A, Q', δ') are said to be *isomorphic* if there exists a bijection $\chi : Q \to Q'$ such that

$$(\chi \times id_{A_\star}) \circ \delta' = \delta \circ (\chi \times id_{A_\star}).$$

Following Katis *et al.* (2002), we shall be dealing with isomorphism classes of Turing automata, while in practice working with representatives.

Because of the discrete nature of the semantics involved, we shall assume that the cardinality of the sets A and Q is at most countably infinite. The role of the anchor as a distinguished interface will be explained later. The transition relation δ can either be considered as a function $Q \times A_\star \to \mathscr{P}(Q \times A_\star)$ or as a function $Q \times (A_\star \times A_\star) \to \mathscr{P}(Q)$, giving rise to a Mealy or Medvedev type automaton, respectively. We shall favour the latter interpretation, and call $\Theta_A = A_\star \times A_\star$ the *input alphabet* for T. Then, as customary in automata theory, the extension of δ to input strings in Θ_A will be denoted by δ too.

By the standard definition, T is *deterministic* if δ is a partial function in the chosen Medvedev sense. In contrast, T is *strongly deterministic* if δ is a partial function in the Mealy sense. Despite using the Medvedev style formalism, we will still say that T has a transition from a to b in state q, resulting in state r, if $((q, a), (r, b)) \in \delta$ (that is, $r \in \delta(q, (a, b))$). According to this philosophy, if A is finite and $|A| = n$, then δ is an $(n+1) \times (n+1)$ matrix, where each entry is a relation over Q. Dually, we can also consider T as an automaton with states $A_\star$ and inputs $Q \times Q$. We shall reflect on this duality shortly.

Example 6.1. Consider the primitive model in Figure 18a with one tape cell in a Turing machine (TM, for short) M holding a symbol a. If M has tape alphabet Γ and states $\{s_1, \ldots, s_n\}$, we represent this cell by the TA given by $C = (A + A, Q, \delta)$, where $A = [n]$, $Q = \Gamma$, and δ is essentially the transition relation of M – see Figure 18b. Recall that the transition relation of M specifies one move of M as a 5-tuple (s_i, a, s_j, b, D), where $a, b \in \Gamma$ and D is a direction, that is, L or R. Spelling this out, if M is in state s_i and its tape head is scanning a cell that holds symbol a, then M changes its state to s_j, rewrites the symbol a to b and its tape head then moves in direction D. Our interpretation δ of one move also takes into account the direction from which the tape head is coming, but this is a trivial issue that has no effect on the computational power of Turing machines. (Indeed, one might consider states (s_i, L) and (s_i, R) in a variant of M.) The main point of this analogy is that it illustrates the dual viewpoint from which states of M are viewed as interfaces (inputs) of C, whereas tape symbols (inputs) of M are in fact states of C.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

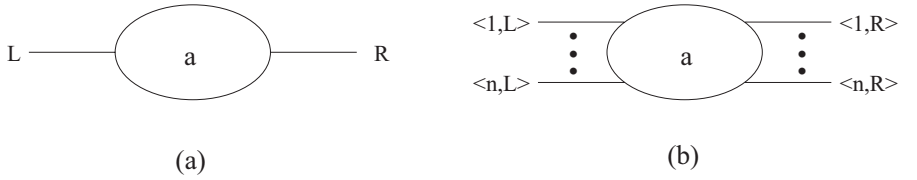


Fig. 18. One tape cell in a Turing machine.

To avoid confusion, we shall sometimes refer to states of M as *global states* and to tape symbols as *local states*.

According to the automaton C, there are no transitions to or from the anchor $\star$. In order to start the computation from somewhere, however, one must also consider an extension C_0 of C in which such transitions also exist. The cell C_0 will have a unique instance in the middle of the two-way infinite array of cells representing M.

We now turn to defining the indexed monoidal algebra $\mathscr{T}$ of Turing automata. In this algebra, morphisms are Turing automata $T : A$. A set-permutation symbol $\rho : A \Rightarrow B$ is interpreted as the relabelling of the interfaces according to the unique bijection $A \to B$ determined by ρ in **Bset** (the elaboration of the details regarding the transition relation is left as an exercise). Clearly, every bijection $\chi : A \to B$ can be interpreted as a relabelling of any TA $T : A$. It is therefore natural and inviting to write $T \cdot \chi : B$ for the automaton so obtained. Even though not all bijections χ in **Bset** are permutations, χ is still a trivial one-state TA $A \to B$ (that is, $\chi : A + B$). Thus, $T \circ \chi$ is going to be meaningful in the algebra $\mathscr{T}$, and its meaning will be identical to $T \cdot \chi$ whenever χ is a permutation.

The tensor of $T : A$ and $T' : B$ having states Q and Q', respectively, is the automaton

$$T \otimes T' = (A + B, Q \times Q', \delta \otimes \delta'),$$

where

$$\delta \otimes \delta' \subseteq ((Q \times Q') \times (A + B)_\star)^2$$

is defined by

$$((q, q'), x), ((r, r'), y)) \in \delta \otimes \delta'$$

if and only if either $q' = r'$ and $((q, x), (r, y)) \in \delta$, or $q = r$ and $((q', x), (r', y)) \in \delta'$. Notice the ambiguity in writing just x, y rather than $\langle x, A\rangle, \langle y, A\rangle$ or $\langle x, B\rangle, \langle y, B\rangle$. The definition, however, applies to the case $x = \star$ and/or $y = \star$, too, so that taking the tensor of T and T' amounts to a selective performance of δ or δ' on $Q \times Q'$. By the same token, self-transitions of $\star$ according to $T \otimes T'$ are the 'union' of such transitions in T and T' (that is, either one by T or one by T' non-deterministically on the product state space, always leaving the other component unchanged).

The operation $\otimes$ can be naturally generalised to an infinite number of operands, provided we lift the declared requirement on the cardinality of the set of states, or make the additional requirement that only a finite number of operand automata have more than a single state. Let $\{T_i : A_i | i \in I\}$ be a family of Turing automata for some finite or countably infinite set I. Then $\otimes_{i\in I} T_i : \sum_{i\in I} A_i$ is the automaton having states $\times_{i\in I} Q_i$ such

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

that its transition function $\otimes_{i\in I}\delta_i$ is the selective performance of one of the δ_i's on $\times_{i\in I}Q_i$ in the above sense.

The identity Turing automaton $\mathbf{1}_A : A+A$ has a single state in which there is a transition from $\langle a,1\rangle$ to $\langle a,2\rangle$ and back for every $a \in A$. There are no transitions to or from the anchor.

The definition of $\updownarrow_A T$ for a TA $T : (A+A)+B$ is complicated but natural, and it holds the key to understanding the TM-like behaviour of this automaton. Intuitively, the definition models the behaviour of loops in flowchart algorithms (Elgot 1975) when implemented in an undirected environment. That is, control enters $\updownarrow_A T$ at an interface $b \in B$, then, after alternating between corresponding interfaces in $A+A$ any number of times, it leaves at another (or the same) interface $b' \in B$. State changes are traced interactively during this process.

Formally, let

$$u = p_1 \dots p_n \in \Theta^*_{(A+A)+B}$$

be a non-empty input string for T, where $p_i = (x_i, y_i)$, $n \geqslant 1$. The string u is said to be *A-alternating* from x_1 to y_n ($u : x_1 \to y_n$, for short) if for every $i \in [n-1]$ there exist $a_i \in A$ and $j \in \{1,2\}$ such that

$$y_i = \langle a_i, j\rangle \text{ and } x_{i+1} = \langle a_i, 2-j\rangle.$$

Then, for every state $q \in Q$ and input $p = (b_1, b_2)$, the transition function $\hat{\delta}$ of $\updownarrow_A T$ is defined by

$$\hat{\delta}(q,p) = \cup(\delta(q,u) | u : b_1 \to b_2 \text{ is } A\text{-alternating}).$$

The specification of $\hat{\delta}$ in terms of matrices is even more elegant. Let us assume that A and B are both finite and contain elements n and m, respectively. We write the square matrix $\delta : 2n+m+1 \times 2n+m+1$ in the form

$$\begin{pmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{pmatrix},$$

where

$$\begin{aligned} L_{11} &: 2n \times 2n \\ L_{12} &: 2n \times m+1 \\ L_{21} &: m+1 \times 2n \\ L_{2,2} &: m+1 \times m+1. \end{aligned}$$

Then

$$\hat{\delta} = L_{22} + L_{21} \odot L_{11}^{\circledast} \odot L_{12}. \tag{1}$$

In the underlying semiring $R = \mathscr{P}(Q \times Q)$, addition $(+)$ is $\cup$ and multiplication $(\cdot)$ is composition of relations. The symbol $\odot$ denotes alternating matrix product, so for any two matrices $V_1, V_2, n \times _$, we have

$$U \odot \begin{pmatrix} V_1 \\ V_2 \end{pmatrix} = U \cdot \begin{pmatrix} V_2 \\ V_1 \end{pmatrix}.$$

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

The operation $^{\circledast}$ is alternating Kleene star in the following sense. For a $2n \times 2n$ square matrix U,

$$U^{\circledast} = \sum_{n \geqslant 0} U^n,$$

where U^0 is the alternate identity matrix

$$\begin{pmatrix} 0 & I_n \\ I_n & 0 \end{pmatrix},$$

and $U^{i+1} = U^i \odot U$.

It is easy to see that the two definitions of $\hat{\delta}$ coincide. The matrix $L_{11}^{\circledast}$ captures sequences of transitions alternating between corresponding interfaces in $A + A$. The Kleene formula (1) for $\hat{\delta}$ is then self-explanatory.

The classical Kleene formula is one of the earliest findings in automata theory. In its original form it was introduced in Kleene (1956) in the construction to convert a finite automaton to a regular expression. The emergence of this formula in the present context is quite natural. We are dealing with graph semantics as purely sequential computations, and Kleene derived exactly the same kind of semantics from the transition graph (albeit directed) of a finite automaton in the form of a regular language (expression).

Those familiar with iteration theories (Bloom and Ésik 1993) will immediately notice that our $^{\circledast}$ is the undirected counterpart of the star operation in matrix iteration theories. See also the example $(\mathbf{Rel}, +)$ in Joyal *et al.* (1996, Section 6), which itself originates from Bloom and Ésik (1993). The star operation is also used in Girard's original formulation of the Geometry of Interaction (Girard 1989); the execution formula, in particular. On the one hand, the connection between matrix iteration theories and Turing automata is not surprising, since the model of the star operation was also Kleene star for regular sets. On the other hand, even if we define Turing automata as directed computational devices by distinguishing between input and output interfaces and requiring that control follows a strict input–output direction, the resulting traced monoidal category will not be an algebraic theory, and thus not a matrix iteration theory. The reason is the different tensor, which complements the cascade product of automata as composition. We shall return to this argument in Section 7.

Example 6.2. Consider the TA $C = ([n] + [n], \Gamma, \delta)$ described in Example 6.1 as one tape cell of a TM M, and construct the TA

$$C^{\infty} = \otimes_{k \in Z} C_k : \sum_{k \in Z} ([n] + [n]),$$

where Z is the set of integers and $C_k = C$ whenever $k \neq 0$. The automaton C_0 is an extension of C by appropriate transitions involving the anchor. Finally, we define

$$T = \updownarrow_Z (C^{\infty} \cdot \rho) : \varnothing, \tag{2}$$

where ρ is the composite of the restructuring bijection

$$\sum_{k \in Z} ([n] + [n]) \to \sum_{k \in Z} [n] + \sum_{k \in Z} [n]$$

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

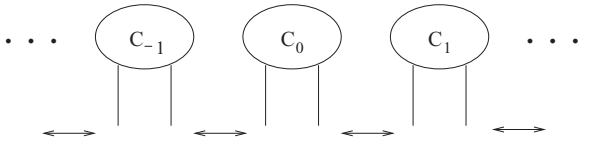


Fig. 19. A TM as a TA.

and the left L-shift

$$\langle\langle i, j\rangle, L\rangle \mapsto \langle\langle i, j-1\rangle, L\rangle$$
$$\langle\langle i, j\rangle, R\rangle \mapsto \langle\langle i, j\rangle, R\rangle.$$

See Figure 19.

The automaton T could be viewed as a model of the whole TM M as a TA. In T, the anchor has taken over the role of an initial global state, and final states in M have been replaced by the requirement that all meaningful computations must return to the anchor. The presence of final states in the definition of Turing machines is specific to these machines being used as language acceptors. With this exclusive goal in mind, every TA derived from a TM can trivially be made deterministic by our Medvedev-style definition. Indeed, whenever the automaton wants to accept (or reject), it will simply erase (blank out) each tape cell before exiting, so rendering the computation automatically deterministic. This will always be feasible, since there were only a finite number of non-blank tape cells (the string to be processed) at the beginning of the computation. Non-halting computations do not matter, for they are not transitions of the TA. A classical deterministic TM, in our language, is derived from a single-tape-cell TA C that is *strongly* deterministic. Obviously, only the strong deterministic property is closed under the indexed monoidal operations, which explains this anomaly.

However, there is one caveat with respect to the automaton T in Example 6.2. It has a continuum number of states. Therefore T cannot be a faithful representation of the TM M. In order to overcome this problem, we must adopt a denotational semantics philosophy and consider a sequence T_l, $l \geqslant 1$ of *approximations* of T in which, for every $|m| \geqslant l$, $C_m : [n] \to [n]$ is the trivial one-state TA having no transitions at all. Intuitively, the single state of this empty automaton is the blank, and whenever control reaches C_m, the machine will crash. Then the 'good' TA that we have in mind is the colimit of the TA T_l according to the following vertical structure in $\mathscr{T}$. For automata

$$T = (A, Q, \delta)$$
$$T' = (A, Q', \delta'),$$

and relation $\phi : Q \to Q'$, we write $\phi : T \to T'$ if ϕ is a *homomorphism* of monoidal automata (Bartha 2009), that is,

$$(\phi \times id_{A_\star}) \circ \delta' = \delta \circ (\phi \times id_{A_\star}).$$

Observe that, by definition, the isomorphism of automata follows exactly this vertical structure as a category. Clearly, there are straightforward injective homomorphisms

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

$\phi_l : T_l \to T_{l+1}$ (remember that the blank is present in Γ), and the colimit of the sequence ϕ_l exists as a TA T_∞ having a countably infinite number of states. It is this automaton T_∞ that models the TM M in a faithful manner.

On the analogy of our SDCC category (**Rel**, $\times$) example, the vertical categories $\mathscr{T}_A$ of Turing automata of rank A are also SDCC categories. Hence they are indexed monoidal algebras. We have thus 'discovered' the concept of 2-indexed monoidal algebras, which are the algebraic counterparts of 2-SDCC categories. We shall elaborate on this issue in a separate paper, but for now we will only observe that the algebra $\mathscr{T}$ has much more structure than we could exhibit in this introductory paper.

The formula (2) above suggests that it is often convenient to express the operation trace in terms of a one-to-one correspondence between two arbitrary disjoint subsets of interfaces. Let:

— $T : D$ be a TA;
— A, A' be disjoint equivalent subsets of D with a specified bijection $\chi : A \to A'$; and
— $B = D \setminus (A \cup A')$.

Then $\updownarrow_\chi T$ is the TA $\updownarrow_A T \cdot \rho_\chi$, where $\rho_\chi : D \to A + A + B$ is the bijection determined by χ in the obvious way. It is easy to verify that $\updownarrow_\chi$ is compatible with relabelling, that is, if

$$\omega : A \to C$$
$$\omega' : A' \to C'$$
$$\psi : C \to C'$$

are bijections for some disjoint sets C, C' that are also disjoint from B such that $\omega \circ \psi = \chi \circ \omega'$, then

$$\updownarrow_\chi T = \updownarrow_\psi T \cdot (\omega \cup \omega' \cup id_B).$$

In particular, $\updownarrow_\chi T = \updownarrow_{\chi^{-1}} T$. In this way, the concept of a χ-*alternating* input string for T is the obvious generalisation of the original A-alternating one.

Lemma 6.1. Let $T = (A \cup A' \cup B, Q, \delta)$ be a TA, and $\chi : A \to A'$ be a one-to-one correspondence as above. For every $b_1, b_2 \in B$ and $q_1, q_2 \in Q$,

$$((q_1, b_1), (q_2, b_2)) \in \delta_\chi \text{ iff } ((q_1, b_1), (q_2, b_2)) \in \delta_{\bar{\chi}},$$

for a restriction $\bar{\chi}$ of χ to a finite subset of A, where δ_χ and $\delta_{\bar{\chi}}$ are the transition relations of $\updownarrow_\chi T$ and $\updownarrow_{\bar{\chi}} T$, respectively.

Proof. The statement follows immediately from the definition of trace in $\mathscr{T}$, which describes a finite process. □

In terms of Turing machines, Lemma 6.1 expresses the trivial fact that each concrete halting run of a TM affects a finite number of tape cells only.

Lemma 6.2. Let $T = (A \cup A' \cup B, Q, \delta)$ be a TA and $\chi : A \to A'$ be a one-to-one correspondence as in Lemma 6.1. Furthermore, we assume that

$$A = A_1 \cup A_2$$
$$A' = A'_1 \cup A'_2$$

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

such that

$$A_1 \cap A_2 = \varnothing$$
$$\chi(A_i) = A_i'.$$

Then

$$\updownarrow_\chi T = \updownarrow_{\chi_2} (\updownarrow_{\chi_1} T)$$

for the restrictions $\chi_i : A_i \to A_i'$ of χ to A_i.

Proof. Let δ_χ, δ_{χ_1} and δ_{χ_2/χ_1} denote the transition functions of $\updownarrow_\chi T$, $\updownarrow_{\chi_1} T$ and $\updownarrow_{\chi_2} (\updownarrow_{\chi_1} T)$, respectively. By definition,

$$\delta_{\chi_2/\chi_1} \subseteq \delta_\chi,$$

therefore we only need to prove that for every state $q \in Q$ and input $(b_1, b_2) \in \Theta_B$,

$$\delta_\chi(q, (b_1, b_2)) \subseteq \delta_{\chi_2/\chi_1}(q, (b_1, b_2)).$$

By Lemma 6.1, we can assume that A is finite. Then, using an appropriate induction argument, we can also assume that $A_2 = \{a\}$ is a singleton.

We use the denotation $a' = \chi(a)$, and let $u = p_1 \dots p_n$ be a χ-alternating input string for T from b_1 to b_2 with $p_i = (x_i, y_i)$. We isolate those indices $i_j \in [n]$, with $j \in [m]$ for which $y_{i_j} = a$ or $y_{i_j} = a'$, and concentrate on the substrings $u_j = p_{i_j+1} \dots p_{i_{j+1}}$, $j \in [m-1]$, together with the prefix $u_0 = p_0 \dots p_{i_1}$ and the suffix $u_m = p_{i_m+1} \dots p_n$. Clearly, each $u_j : \bar{x}_j \to \bar{y}_j$, with $0 \leqslant j \leqslant m$, is a χ_1-alternating input string for T, where

$$\bar{x}_j = \begin{cases} b_1 & \text{if } j = 0 \\ x_{i_j+1} & \text{if } j \geqslant 1, \end{cases} \quad \text{and} \quad \bar{y}_j = \begin{cases} b_2 & \text{if } j = m \\ y_{i_{j+1}} & \text{if } j < m. \end{cases}$$

Similarly, the string $\bar{u} = (\bar{x}_0, \bar{y}_0) \dots (\bar{x}_m, \bar{y}_m)$ is χ_2-alternating for $\updownarrow_{\chi_1} T$ from b_1 to b_2.

Now let $q' \in \delta_\chi(q, (b_1, b_2))$. By definition, there exists a χ-alternating input string $u = p_1 \dots p_n$ for T from b_1 to b_2 such that $q' \in \delta(q, u)$. Let u be as above. Breaking this string down to the substrings $u_0, \dots u_m$, there exist states $q = q_0, q_1, \dots, q_{m+1} = q'$ for which $q_j \in \delta(q_{j-1}, u_{j-1})$, with $j \in [m+1]$. Again by definition, this implies

$$q_j \in \delta_{\chi_1}(q_{j-1}, (\bar{x}_{i-1}, \bar{y}_{i-1}))$$

for each $j \in [m+1]$, therefore $q' \in \delta_{\chi_2/\chi_1}(q, (b_1, b_2))$. □

Theorem 6.3. The algebra $\mathscr{T}$ of Turing automata is indexed monoidal.

Proof. At this point we can capitalise to a great extent on the simplicity of the axioms in IM. Indeed, with the exceptions of vanishing (I7) and trace swapping (I9), each of these axioms holds naturally true in $\mathscr{T}$. However, (I7) and (I9) follow immediately from Lemma 6.2. Observe that, analogously to the circuit model in (Katis *et al.* 2002), superposing (I8) holds up to isomorphism of automata only, which is why we had to work with isomorphism classes of automata in the algebra $\mathscr{T}$. □

Finally, we will explain the role of the anchor $\star$. We did not want all Turing automata of rank $I = \varnothing$ to have no transitions at all, like the automata $\perp_A = \updownarrow_A \mathbf{1}_A$, which all

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

coincide, having a unique state. (Consequently, the canonically traced monoidal category $\mathscr{ST}$ of Turing automata also satisfies the additional 'scheme' axiom $\uparrow_A 1_A = 1_I$, which is quite natural.) The anchor is a fixed interface that is not supposed to be interconnected with any other, so that automata of rank I might still have transitions from $\star$ to $\star$.

Now let us return to the traced monoidal category $\mathscr{B} = (\mathbf{Rel}, +)$ analysed in Joyal *et al.* (1996, Section 6). Observe that the Kleene formula (1) in its non-alternating form defines trace in that category too. This is evident, because the restriction of $\mathscr{B}$ to finite sets is equivalent to the matrix iteration theory of bit matrices. The CC category $Int(\mathscr{B})$ was characterised in technical terms in Joyal *et al.* (1996). Now we can show the intuitive meaning of that structure too. Indeed, it is not difficult to see that the restriction of $Int(\mathscr{B})$ to its self-dual objects (A, A) is isomorphic to the subcategory of $\mathscr{ST}$ consisting of single-state Turing automata with no transitions to or from the anchor. We shall be even more explicit about this observation in Section 7.

The idea of working with a multiplicative and an additive tensor (that is, $\times$ and $+$) at the same time leads directly to the model of quantum Turing automata. The underlying category in this model is changed from sets and relations to finite dimensional Hilbert spaces with linear maps between them. The additive tensor is orthogonal sum, while the multiplicative one is tensor product of Hilbert spaces. Linear maps, however, must be confined to *unitary* ones in order to make the additive style trace operation work. Quantum Turing automata were introduced along these lines in Bartha (2012). In the current paper we present a simpler, but still interesting model, which targets electronic switching at the molecular level. This model has been studied for over a decade now on the grounds of matching theory using the name soliton automaton, *cf.* Bartha and Krész (2010).

Example 6.3. The *n-ary atomic switch* is the Turing automaton $\mathscr{A}_n : [n]$ $(n \geqslant 1)$ having states $[n]$, such that

$$\delta = \{((\mathbf{i}, j), (\mathbf{j}, i)) \mid 1 \leqslant i \neq j \leqslant n\} \cup \{((\mathbf{i}, i), (\mathbf{j}, j)) \mid 1 \leqslant i \neq j \leqslant n\}$$

$$\cup \{((\mathbf{i}, *), (\mathbf{i}, j)), ((\mathbf{i}, j), (\mathbf{i}, *)), ((\mathbf{i}, *), (\mathbf{i}, *)) \mid i, j \in [n]\}.$$

For better readability, states, indicating a selected edge in an n-star graph, are written in boldface. In addition, if $n = 1$, then $((\mathbf{1}, 1), (\mathbf{1}, 1)) \in \delta$.

Heuristically, the n-ary atomic switch captures the behaviour of an atom in a molecule having n chemical bonds to neighbouring atoms. Among these bonds, exactly one is double, and referred to as the *positive* edge in the underlying star graph. The mechanism of switching is then clear by the definition above. The active ingredient (control) in this process is called the *soliton*, which is a form of energy travelling in small packets through chains of alternating single and double bonds within the molecule, and causing the affected bonds to be flipped from single to double and *vice versa*. See Davidov (1985) for the physico-chemical details, and Dassow and Jürgensen (1990), Bartha and Krész (2003) and Bartha and Krész (2006) for the corresponding mathematical model. Note that by our definition above, whenever the soliton enters an atom with a unique chemical bond

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press



Fig. 20. The von Neumann machine.

(which must be double since $n = 1$), it bounces back immediately, producing no state change.

Let D be a non-empty set of data such that $|D|$ is at most countably infinite. The indexed monoidal algebra $D\text{-}dil\mathscr{T}$ of *D-flow* Turing automata is defined as follows:

— Morphisms of rank A are Turing automata $T : D \times A$.
— Each permutation symbol $\rho : A \Rightarrow B$ is interpreted as a bijection (relabelling) $D \times A \to D \times B$, which is basically $\rho_{\mathscr{T}}$ performed on blocks of size D in parallel.
— The tensor is adopted from $\mathscr{T}$ (assuming the identification of $D \times (A + B)$ with $D \times A + D \times B$), $(\updownarrow_A T)_{D\text{-}dil\,T} = \updownarrow_{D\times A} T$, and the identities $\mathbf{1}_A$ are the identities $\mathbf{1}_{D\times A}$ in $\mathscr{T}$.

The notation *D-dil* is taken from Arnold and Dauchet (1978), where the magmoid (single-sorted monoidal category) $k\text{-}dil\mathscr{M}$ was introduced for integer k and magmoid $\mathscr{M}$ along these lines. Intuitively, a D-flow Turing automaton is a data-flow machine in which data in D are passed along with each transition. Data appear disguised, however, as 'cloned' interfaces. In other words, computation remains unary since, technically, the control still does not carry information. Notice that the anchor does not emit or receive any data. By virtue of Lemma 6.2 and Theorem 6.3, the structure $D\text{-}dil\mathscr{T}$ of D-flow Turing automata is an indexed monoidal algebra.

Consider, for example, the scheme N of the classical von Neumann computer in Figure 20 as a data-flow architecture. It consists of two interconnected single-sorted D-flow Turing automata: the processor $P : 2$ and the memory $M : 1$. The processor is a real finite-state automaton, having state components like registers, the instruction counter, the PSW, and so on. The transitions of P are very complex. On the other hand, M has (practically) infinite states, but its transitions are straightforward. The set D consists of all pieces of information (data, control and/or address) that can be transmitted along the bus line between P and M in either direction. The operation of N need not be explained, and it is clearly that of a D-flow Turing automaton. It is a very important point, however, that the machine as a TA can do as much as we want in one step, that is, from the time control enters port 1 of P until it leaves at the same port. For example, it can execute one machine instruction stored in the memory, or even a whole program stored there. In other words, the semantics is delay-free. In present-day digital computers, the desired semantics is achieved by limiting the scope of what the machine can do in one step through the introduction of clock cycles and delay. Theoretically speaking, undirected trace is turned into directed feedback with delay (or, using everyday language, recursion is transformed into a loop), and computations become inevitably directed in a rigid way. According to the original scheme N, however, they need not be, and they could even remain universal.

The problem is that we are not able to build up either the processor or the memory as a deterministic Turing graph machine from concrete physical atomic TA. The logical gates and flip-flops used in sequential circuits to build up the processor and the memory are not

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

Turing automata. According to the resulting common digital architecture, the computer itself is a pair of interconnected Mealy automata, that is, a synchronous system in the sense of Leiserson (1983). In the characterisation of synchronous systems as monoidal automata (Bartha 1987b; Bartha 1992), the underlying monoidal category is (a subcategory of) $(\mathbf{Set}, \times)$. The tensor of such automata is therefore related to the cartesian product of sets and functions, as opposed to the disjoint union of sets and relations according to the Turing automaton model. The composition of automata is cascade product, and feedback is delayed. The monoidal structure of synchronous systems obeys exactly the traced monoidal (alias scheme) axioms, except for yanking, which does not hold in these categories. The trace (feedback) of the symmetry $c_{U,U}$ is the unit delay, or register ∇_U. See Section 7 for further explanation.

Regarding the speed of computation, the impact of introducing a fixed delay is significant. It is well known in complexity theory that the time complexity of a TM is comparable to that of an equivalent RAM (random access machine, that is, von Neumann machine) program. The reason is that even though the processor and the memory are not TA-like, control will only stay for a constant number of clock cycles in these components every time it enters either of them during the execution of a program. In other words, complexity can be measured in the number n of times control passes through the bus, which is a TA-like bidirectional interconnection. We must, however, multiply n by at least the fixed length t of one clock cycle. No matter how small t is nowadays, it is still very large in contrast to the speed by which particles could ideally travel in a delay-free TA-like quantum computer as described in Bartha (2012).

The soliton automaton model, introduced in Example 6.3 through several continuations, makes an attempt to at least partially solve the problem of implementing electronic switching by Turing automata built up from atomic components.

Example 6.3 (continued). The n-ary atomic *alternating* switch $\mathscr{A}_n^2$ augments the ordinary n-ary atomic switch by the passing of digital information in the following way. Control from a negative interface (that is, one not covered by the unique positive edge) can only take 0 for input and emits 1 for output. (Remember that in the meantime the positive edge is switched from the output side to the input side.) Conversely, control from a positive interface can only take 1 for input and emits 0 for output. Transitions to and from the anchor are as in the corresponding $2n$-ary switch.

For the rest of this paper, the alphabet Σ will be single-sorted, that is, $\Sigma = (\Sigma_n \mid n \geqslant 0)$.

Definition 6.2. A *D-flow Turing graph machine* over Σ is a triple $M = (G, D, \Omega)$, where G is a Σ-graph, D is a non-empty set (at most countably infinite), and Ω is an interpretation of Σ in $\mathscr{T}$ under which the single sort of Σ is mapped into D. Equivalently, Ω is an interpretation in $D\text{-}dil\,\mathscr{T}$ that maps sort '1' to object $\{1\}$.

Intuitively, machine M comes with an underlying graph G that has a D-flow Turing automaton sitting in each of its internal vertices. The operation of M as a complex Turing automaton is uniquely determined by the given interpretation according to the homomorphism Ω. The classical (bounded) Turing machine concept can again be recaptured by taking $\Sigma = \Sigma_2 = \{c\}$, where c stands for 'tape cell'. A Turing machine TM

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

is transformed into a D-flow Turing graph machine M whose underlying graph is a linear array of cell vertices with the following interpretation $T_c : 2$ of c. The states of T_c are the tape symbols (local states) of TM, and, by way of duality, the elements of D are the global states of TM. The transition relation of TM translates directly and naturally into that of M, using duality. The only shortcoming of this analogy is the finiteness of the underlying graph G. Notice that it is not possible to specify the TA $\updownarrow_Z C^\infty$ in equation (2) as a Turing graph machine, since the graph structure as generated from the ranked alphabet Σ is necessarily finite. To extend the syntax to infinite graphs, we may consider the standard subgraph relationship as a partial order and make it closed for taking limits. Then the well-known denotational technique can be adopted to extend the semantics Ω to infinite graphs. Of course, we are normally interested in infinite graphs of a certain 'regular' pattern, and these graphs could be generated by appropriate grammars. See, for example, Bauderon and Courcelle (1987) and Engelfriet and Vereijken (1997). Semantics could then be carried over to such grammars algebraically.

Example 6.3 (continued further). Let Σ be the ranked alphabet consisting of a single symbol c_n for each rank $n \geqslant 1$. A *pre-soliton automaton* is a Turing graph machine

$$S = (G, \{0, 1\}, \Omega),$$

where Ω is the fixed interpretation that sends each symbol c_n into the n-ary atomic alternating switch $\mathscr{A}_n^2$. Since the interpretation is fixed, we shall identify each pre-soliton automaton with its underlying graph. Moreover, since $\mathscr{A}_n^2$ is circularly symmetric, we do not need to order the ports (degrees) of the internal vertices. Thus, G is an ordinary open undirected graph as described in Example 4.1.

Let q be a state of graph G. By definition, each internal edge $e \in E$ is either *consistent* with respect to q, meaning that e has the same sign (positive or negative) viewed from its two internal endpoints, or *inconsistent* if this is not the case (note that a looping edge is always negative if it is consistent). A *soliton walk* from interface i to interface j ($i, j \in [n] + \{*\}$) is a transition of G from i to j in state q according to the standard behaviour of G as a Turing automaton. It is now easy to verify that this definition of soliton walks coincides with the original one given in Dassow and Jürgensen (1990), provided q is a *perfect internal matching* (Lovász and Plummer 1986; Bartha and Krész 2003) of G. In our language, a perfect internal matching is a state q of G such that every edge of G is consistent with respect to q, and the positive edges determine a matching through which the internal vertices are all covered. Indeed, the definition of $\mathscr{A}_n^2$ implies that the soliton can only traverse consistent edges in an alternating positive–negative fashion. A new feature of this model is a soliton walk from the anchor, which must return to the anchor if q is a perfect internal matching, and in that case it defines a closed alternating walk (for example, an alternating cycle).

At this point we will stop elaborating on soliton automata – see the recent study Bartha and Krész (2011) for further details. The key observation enabling the restriction of the states of pre-soliton automata to perfect internal matchings is the Gallai–Edmonds Structure Theorem (Lovász and Plummer 1986), which is well known in matching theory. On the basis of this theorem, the Gallai–Edmonds algebra of graphs having a perfect

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

internal matching was worked out in Bartha and Gombás (1991) as the homomorphic image of the indexed monoidal algebra $\mathscr{G}$. The IMA of soliton automata then turns out to be an appropriate quotient of the Gallai–Edmonds algebra. See Krész (2007) and Krész (2008) for more information on soliton automata.

Finally, we return to our original dilemma of reversible *versus* irreversible computations and explicitly define the 'reverse' of a Turing automaton $T = (A, Q, \delta)$ simply as $T^R = (A, Q, \delta^{-1})$. This definition only makes T actually reversible if $\delta : A \times Q \to A \times Q$ is a partial injection. It is known (*cf.* Abramsky *et al.* (2002), Lutz and Derby (1982) and Bennett (1973)) that Turing machines with such a restricted capability still have sufficient power to remain universal. However, the effective construction of a reversible universal Turing graph machine poses an enormous challenge. The soliton automaton model described above is an interesting try, but, unfortunately, it falls short of being universal even in terms of designing individual *ad hoc* machines.

7. Related work

The model of Turing automata has grown out of the study of general monoidal automata (Bartha 1992; Bartha 2008; Bartha 2009), also known as circuits (Katis *et al.* 2002). The category structure of circuits comes very close to that of Turing automata, but it is not a traced monoidal category.

According to Katis *et al.* (2002), a *category with feedback* is a monoidal category $\mathscr{C}$ equipped with a feedback operation $\uparrow_{A,B}^{U} : \mathscr{C}(U \otimes A, U \otimes B) \to \mathscr{C}(A, B)$, which satisfies all the requirements for $Tr_{A,B}^{U}$ in Definition 2.1, except for the yanking axiom. Furthermore, when the morphism $g : U \to V$ is an isomorphism, only a weaker form of sliding holds.

For an arbitrary monoidal category $\mathscr{C}$, a *$\mathscr{C}$-automaton* $A \to B$ is a pair (U, α), where U is an object and $\alpha : U \otimes A \to U \otimes B$ is a morphism in $\mathscr{C}$. The pair (U, α) typically models a Mealy automaton, for example, a sequential circuit. Reflecting this interpretation, the object U is called the *state component*, while α is the *combinational logic* of (U, α). The collection of $\mathscr{C}$-automata can be given the structure of a category $Aut_{\mathscr{C}}$ equipped with a tensor as follows. Objects, and the tensor of them, are as in $\mathscr{C}$. Furthermore,

$$\begin{aligned} 1_A &= (I, 1_A) \\ (U, \alpha) \circ (V, \beta) &= (U \otimes V, (c_{U,V} \otimes 1_A) \circ (1_V \otimes \alpha) \circ (c_{V,U} \otimes 1_B) \circ (1_U \otimes \beta)) \\ (U, \alpha) \otimes (V, \beta) &= (U \otimes V, (1_U \otimes c_{V,A} + 1_C) \circ (\alpha \otimes \beta) \circ (1_U + c_{B,V} + 1_D)). \end{aligned}$$

See Figure 21. The type of composition shown in Figure 21 is commonly known as the *cascade product* of automata. The category $\mathscr{C}$ is embedded into $Aut_{\mathscr{C}}$ by the functor $A \mapsto A$, $\alpha \mapsto (I, \alpha)$. Feedback is defined in $Aut_{\mathscr{C}}$ as follows:

$$\uparrow_{A,B}^{V} (U, \alpha) = (U \otimes V, \alpha) \quad \text{where } \alpha : U \otimes (V \otimes A) \to U \otimes (V \otimes B).$$

Two $\mathscr{C}$-automata $(U, \alpha), (V, \beta) : A \to B$ are *isomorphic* if there exists an isomorphism $\gamma : U \to V$ in $\mathscr{C}$ such that

$$(\gamma \otimes 1_A) \circ \beta = \alpha \circ (\gamma \otimes 1_B).$$

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

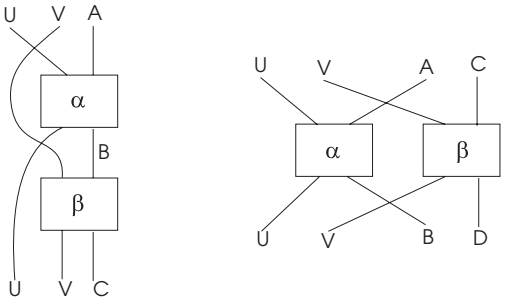


Fig. 21. Composition and tensor in $Aut_{\mathscr{C}}$

Isomorphism classes of $\mathscr{C}$-automata are called *circuits*, and $Circ(\mathscr{C})$ is the quotient of $Aut_{\mathscr{C}}$ by this isomorphism. As proved in Katis *et al.* (2002), the tensor (of isomorphism classes of $\mathscr{C}$-automata) makes $Circ(\mathscr{C})$ a monoidal category with the symmetry adopted from M. Moreover, $Circ(\mathscr{C})$ is a category with feedback that is freely generated by $\mathscr{C}$. In our intuitive interpretation, feedback in $Circ(\mathscr{C})$ turns interfaces (input/output channels) into registers (state components). Note that, by the Mealy automaton interpretation of (U, α), the duality principle with respect to input *versus* state is not in effect. Inputs and outputs are signals 0,1, and states are flip-flops/registers. It is also important to see that the Int construction does not work for monoidal categories with a delayed feedback. The unit delay cannot be reversed into a 'negative' one. Designing a reversible computation model strictly on the principle of Mealy automata as input–output devices is therefore impossible.

As a comparison, we will now build up the indexed monoidal algebra of Turing automata from the appropriate monoidal ones, which we call *directed* Turing automata. Feedback (trace), as well as tensor, follows an entirely different philosophy in directed Turing automata, even though composition between them is still the old cascade product. Trace only cuts interfaces in a Turing automaton without creating registers from them. The state component of the resulting automaton remains the same object (set), but the transition relation (combinational logic) becomes a lot more complicated. Most importantly, though, trace complements an additional additive tensor, rather than $\otimes$.

Formally, let $\mathscr{C}$ have a further *additive tensor* $\oplus$ such that $\otimes$ distributes over $\oplus$. In our example category of sets and relations, $\otimes = \times$ and $\oplus = +$. Then, using the simplified strict formalism, the Turing tensor of automata $(U, \alpha) : A \to B$ and $(V, \beta) : C \to D$ is

$$(U, \alpha) \boxtimes (V, \beta) = (U \otimes V, \gamma) : A \oplus C \to B \oplus D,$$

where $\gamma \simeq \gamma_1 \oplus \gamma_2$ is given by

$$\gamma_1 = (c_{U,V} \otimes 1_A) \circ (1_V \otimes \alpha) \circ (c_{V,U} \otimes 1_B) : (U \otimes V) \otimes A \to (U \otimes V) \otimes B,$$
$$\gamma_2 = 1_U \otimes \beta : (U \otimes V) \otimes C \to (U \otimes V) \otimes D.$$

The natural isomorphism $\simeq$ above is left-distributivity. The intuitive idea behind the Turing tensor $\boxtimes$ should be clear: it is the selective performance of either α or β over $U \otimes V$. The duality principle with respect to input and state does apply, so the whole

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

automaton concept requires a completely different intuitive understanding from that of Mealy automata. The anchor as a distinguished interface is specific to the relational Turing automaton model, and is missing from the present general discussion.

It is not difficult to show that the Turing tensor, too, turns the category of (isomorphism classes of) $\mathscr{C}$-automata into a monoidal one $Tur(\mathscr{C})$. Bear in mind, however, that the object structure in $Tur(\mathscr{C})$ has $\oplus$ for tensor, rather than $\otimes$ as in $Circ(\mathscr{C})$. Now let us assume that $\mathscr{C}$ admits an additive trace, too, by which $(\mathscr{C}, \oplus)$ is a traced monoidal category. Then, under certain natural conditions, this trace can be carried over to the category $(Tur(\mathscr{C}), \boxtimes)$, so that it becomes traced monoidal as well. This is definitely the case with our example category $\mathscr{C} = (\mathbf{Rel}, \times, +)$, so the traced monoidal category $\vec{\mathscr{T}}$ of directed Turing automata is well established. As another example, directed quantum Turing automata were built up along these lines in Bartha (2012) using the base category of finite dimensional Hilbert spaces and isometries between them, with the two tensors being tensor product and orthogonal sum. In the quantum example, the additive trace relies on the Moore–Penrose generalised inverse of linear operators – *cf.* Roman (2005).

Given the traced monoidal category $(Tur(\mathscr{C}), \boxtimes)$, we can transform it into the CC category $Int(Tur(\mathscr{C}))$, restrict the scope of this category to self-dual objects (A, A) and convert the resulting SDCC category to an IMA. It is in exactly this way that the IMA $\mathscr{T}$ of (undirected) Turing automata can be derived from the traced monoidal category $\vec{\mathscr{T}}$ of directed Turing automata. In Section 6, we deliberately chose an accelerated and more intuitive direct way of introducing the algebra $\mathscr{T}$.

There was a prior attempt by Hines (Hines 2003) to use the Int construction to derive two-way finite state automata from ordinary ones, and bounded Turing machines from finite state Mealy automata. For finite state automata, the starting point is that the transition function of such a machine can be modelled as a homomorphism $\Sigma^* \to End(Q)$, where Σ is the input alphabet, Q is the set of states, and $End(Q)$ is the endomorphism monoid of the object Q in the monoidal category $(\mathbf{Rel}, +)$. In fact, this was not a new idea, since finite state automata (deterministic or not) had been described before and in a more general way by the concept of presentations in matrix iteration theories (Bloom and Ésik 1993). The basic observation made by Hines is that the transition function of a two-way automaton can be modelled by a homomorphism $\Sigma^* \to End(Q_l, Q_r)$ with respect to the CC category $Int(\mathbf{Rel}, +)$, where Q_l and Q_r are the left-moving and right-moving states, respectively. This idea is indeed somewhat reminiscent of our construction of undirected Turing automata from directed ones as outlined above in the trivial special case when all automata have a single state. Remember the duality by which (global) states of machines are interfaces, and inputs (local states) are 'the' states in Turing automata. In our language, literally, Hines' observation constructs an individual Turing graph machine for each input string $w \in \Sigma^*$, which consists of a linear array of cell vertices. Each cell is labelled by a letter of w in a left-to-right way. The interpretation of any such a cell is then a single-state Turing automaton, since the machine does not overwrite the symbol stored in the cell.

Moving on to Mealy automata and Turing machines, Hines did not observe the category of monoidal automata with the cascade product as composition. Rather, he introduced his own $Comp$ construction, which produces a *graded* monoidal category from a monoidal category $\mathscr{V}$; that is, $Comp(\mathscr{V})$ is a functor from a monoidal category to the monoid

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

$(N, +, 0)$, considered as a one-object category – see Hines (2003) for the details. Intuitively, the aim of the *Comp* construction is to reconstruct the behaviour of a Mealy/Turing machine on an $(n + m)$-long tape from its behaviour on an n-long and an m-long tape. Again, the idea is similar to what we have accomplished with the algebra $\mathscr{T}$, but the structure involved is much less sophisticated. In his summary, Hines himself admits that 'In particular, we would like to be able to say that $Int(Comp(\mathscr{V}))$ is isomorphic to $Comp(Int(\mathscr{V}))$, but giving this a precise meaning (or indeed, any meaning) appears to require substantial algebraic input'. Indeed, $Comp(V)$ is not even a category, let alone monoidal or traced, therefore $Int(Comp(\mathscr{V}))$ is not meaningful.

8. Conclusion

We have provided a theoretical foundation for the study of Turing machines. In the first half of the paper we established the categorical framework for this foundation. The basic underlying structure, called indexed monoidal algebra, was introduced through a small number of equational axioms, and it was shown that the category of such algebras is equivalent to that of self-dual compact closed categories. A graphical language for indexed monoidal algebras was worked out and a coherence theorem was proved to justify this graphical language.

In the second half of the paper we defined the indexed monoidal algebra of Turing automata, and showed how the classical Turing machine concept can be recaptured by this definition. Turing graph machines were defined as interpretations of graphs in the algebra of Turing automata. These machines have been generalised to D-flow Turing graph machines, in which data from a set D are passed along with each transition. We have pointed out that, at least at the high level of design, a von Neumann computer architecture is a simple D-flow Turing graph machine. Finally, we have presented the soliton automaton example for switching at the molecular level by fully reversible binary Turing graph machines.

9. Acknowledgment

The author is indebted to the anonymous referee, whose helpful comments have greatly contributed to an improvement in the presentation of the paper.

References

Abramsky, S. (2005a) A structural approach to reversible computation. *Theoretical Computer Science* **347** 441–464.

Abramsky, S. (2005b) Abstract Scalars, Loops, and Free Traced and Strongly Compact Closed Categories. In: CALCO 2005. *Springer-Verlag Lecture Notes in Computer Science* **3629** 1–31.

Abramsky, S. and Coecke, B. (2004) A categorical semantics of quantum protocols. In: *Proceedings of the 19th Annual Symposium on Logic in Computer Science, LICS 2004*, IEEE Computer Society Press 415–425.

Abramsky, S., Haghverdi, E. and Scott, P. (2002) Geometry of Interaction and Linear Combinatory Algebras. *Mathematical Structures in Computer Science* **12** 625–665.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

Arnold, A. and Dauchet, M. (1978-79) Théorie des magmoïdes. In: *RAIRO – Theoretical Informatics and Applications* **12** 235–257 and **13** 135–154.

Bartha, M. (1987a) A finite axiomatization of flowchart schemes. *Acta Cybernetica* **8** (2) 203–217. (Also available online at `http://www.cs.mun.ca/~bartha/linked/flow.pdf`.)

Bartha, M. (1987b) An equational axiomatization of systolic systems. *Theoretical Computer Science* **55** 265–289.

Bartha, M. (1992) An algebraic model of synchronous systems. *Information and Computation* **97** 97–131.

Bartha, M. (2008) Simulation equivalence of automata and circuits. In: Local Proceedings, 12th International Conference on Automata and Formal Languages, Balatonfüred, Hungary 86–99. (Available online at `http://www.cs.mun.ca/~bartha/linked/afl08.pdf`.)

Bartha, M. (2009) Equivalence relations of Mealy automata. In: Local Proceedings, First Workshop on Non-Classical Models of Automata and Applications, Wroclaw, Poland 31–45. (Available online at `http://www.cs.mun.ca/~bartha/linked/wroc.pdf`.)

Bartha, M. (2012) Quantum Turing automata. Proceedings, 8th International Workshop on Developments of Computational Models (DCM 2012). *Electronic Proceedings in Theoretical Computer Science* (in press).

Bartha, M. and Gombás, É. (1991) The Gallai–Edmonds algebra of graphs. Technical Report 9105, Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada. (Available online at `http://www.cs.mun.ca/~bartha/linked/g-e.pdf`.)

Bartha, M. and Jürgensen, H. (1989) Characterizing finite undirected multigraphs as indexed algebras. Technical Report 252, Department of Computer Science, The University of Western Ontario, London, Ontario, Canada. (Available online at `http://www.cs.mun.ca/~bartha/linked/ind.pdf`.)

Bartha, M. and Krész, M. (2003) Structuring the elementary components of graphs having a perfect internal matching. *Theoretical Computer Science* **299** 179–210.

Bartha, M. and Krész, M. (2006) Deterministic soliton graphs. *Informatica* **30** 281–288.

Bartha, M. and Krész, M. (2010) Soliton circuits and network-based automata: review and perspectives. In: Martín-Vide, C. (ed.) *Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory* **2**, Scientific Applications of Language Methods, Imperial College Press 585–631.

Bartha, M. and Krész, M. (2011) Molecular switching by Turing automata. In: Local Proceedings, Third Workshop on Non-Classical Models of Automata and Applications, Milan, Italy 31–45. (Available online at `http://www.cs.mun.ca/~bartha/linked/mol.pdf`.)

Bauderon, M. and Courcelle, B. (1987) Graph expressions and graph rewritings. *Mathematical Systems Theory* **20** 83–127.

Bennett, C. H. (1973) Logical reversibility of computation. *IBM Journal of Research and Development* **17** (6) 525–532.

Bloom, S. L. and Ésik, Z. (1985) Axiomatizing schemes and their behaviors. *Journal of Computer and System Sciences* **31** 375–393.

Bloom, S. L. and Ésik, Z. (1993) *Iteration Theories: The Equational Logic of Iterative Processes*, Springer-Verlag.

Burstall, R. M., Goguen, J. A. and Tarlecki, A. (1989) Some fundamental tools for the semantics of computation. Part 3: Indexed categories. Report ECS-LFCS-89-90, Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh, United Kingdom.

Căzănescu, V. E. and Ştefănescu, Gh. (1990) Towards a new algebraic foundation of flowchart scheme theory. *Fundamenta Informaticae* **13** 171–210. (Also appeared as: *INCREST Preprint Series in Mathematics* **43**, Bucharest, Romania 1987.)

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

Dassow, J. and Jürgensen, H. (1990) Soliton automata. *Journal of Computer and System Sciences* **40** 154–181.

Davidov, A. S. (1985) *Solitons in Molecular Systems*, Reidel, Dordrecht.

D'Hondt, E. and Panangaden, P. (2006) Quantum weakest preconditions. *Mathematical Structures in Computer Science* **16** (3) 429–451.

Elgot, C. C. (1975) Monadic computations and iterative algebraic theories. In: Rose, H. E. and Shepherdson, J. C. (eds.) *Logic Colloquium 1973*, Studies in Logic and the Foundations of Mathematics **80**, North Holland 175–230.

Engelfriet, J. and Vereijken, J. J. (1997) Context-free graph grammars and concatenation of graphs. *Acta Informatica* **34** 773–803.

Freyd, P. and Yetter, D. (1992) Coherence theorems via knot theory. *Journal of Pure and Applied Algebra* **78** 49–76.

Girard, J.-Y. (1987) Linear logic. *Theoretical Computer Science* **50** 1–102.

Girard, J.-Y. (1989) Geometry of Interaction I: Interpretation of System F. In: Ferro, R. *et al.* (eds.) *Logic Colloquium'88*, North-Holland 221–260.

Girard, J.-Y. (1990) Geometry of Interaction II: Deadlock-free Algorithms. In: Martin-Lof, P. and Mints, G. (eds.) COLOG-88. *Springer-Verlag Lecture Notes in Computer Science* **417** 76–93.

Hines, P. (2003) A categorical framework for finite state machines. *Mathematical Structures in Computer Science* **13** 451–480.

Hyland, M. (1997) Game semantics. In: Pitts, A. M. and Dybjer, P. (eds.) *Semantics and Logics of Computation*, Cambridge University Press 131–194.

Joyal, A. and Street, R. (1991) The geometry of tensor calculus I. *Advances in Mathematics* **88** (1) 55–112.

Joyal, A., Street, R. and Verity, D. (1996) Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society* **119** 447–468.

Katis, P., Sabadini, N. and Walters, R. F. C. (2002) Feedback, trace, and fixed-point semantics. *Theoretical Informatics and Applications* **36** 181–194.

Kelly, G. M. (1982) *Basic Concepts of Enriched Category Theory*, London Mathematical Society Lecture Notes Series **64**, Cambridge University Press.

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

Kleene, S. C. (1956) Representation of events in nerve nets and finite automata. In: Shannon, C. E. and McCarthy, J. (eds.) *Automata Studies*, Princeton University Press 3–42.

Krész, M. (2007) Graph decomposition and descriptional complexity of soliton automata. *Journal of Automata, Languages and Combinatorics* **12** 237–263.

Krész, M. (2008) Soliton automata with constant external edges. *Information and Computation* **206** 1126–1141.

Lafont, Y. (1989) Interaction nets. In: *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* 95–108.

Landauer, R. (1961) Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development* **5** 183–191.

Leiserson, C. E. and Saxe, J. B. (1983) Optimizing synchronous systems. *Journal of VLSI and computer systems* **1** 41–67.

Lovász, L. and Plummer, M. D. (1986) *Matching Theory*, North Holland.

Lutz, C. and Derby, H. (1982) Janus: A Time-Reversible Language. (Available online at `http://tinyurl.com/janus82`.)

Mac Lane, S. (1963) Natural associativity and commutativity. Rice University Studies **49** 28–46.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press

Mac Lane, S. (1971) *Categories for the Working Mathematician*, Springer-Verlag.
Mac Lane, S. and Paré, R. (1985) Coherence for bicategories and indexed categories. *Journal of Pure and Applied Algebra* **37** 59–80.
Milner, R. (2009) *The Space and Motion of Communicating Agents*, Cambridge University Press.
Roman, S. (2005) *Advanced Linear Algebra*, Springer-Verlag.
Selinger, P. (2009) A survey of graphical languages for monoidal categories. In: Coecke, B. (ed.) New Structures for Physics. *Springer-Verlag Lecture Notes in Physics* **813** 289–355.

https://doi.org/10.1017/S0960129512000096 Published online by Cambridge University Press