

A categorical framework for finite state machines

PETER HINES

*Oxford University Computing Laboratory, Wolfson Building, Parks Road,
Oxford OX1 3QD, England*
Email: Peter.Hines@comlab.ox.ac.uk

Received 10 March 2001; revised 8 September 2002

We provide a consistent way of looking at a range of finite state machines and their algebraic models. Our claim is that the natural representation of transitions of finite state machines is in terms of monoid homomorphisms, and distinct generalisation processes that can be applied to finite state machines correspond to distinct categorical generalisation processes at the level of the algebraic models.

The generalisations we consider are those from deterministic to non-deterministic machines, from one-way to two-way machines, and from read-only machines to read/write machines. Hence the finite state machines we consider, and provide algebraic models for, are (deterministic and non-deterministic) finite state automata, two-way automata, Mealy machines, and bounded Turing machines.

The categorical constructions corresponding to these generalisation processes are, respectively: altering the base category from functions to relations, applying the Geometry of Interaction, or **Int** construction, and a categorical process, which we refer to as the **Comp** construction, that uses the tensor on monoidal categories to construct graded categories.

1. Introduction

We consider three independent generalisation processes that can be applied to finite state machines. These are as follows:

- Deterministic \rightarrow non-deterministic
- One-way \rightarrow two-way
- Read-only \rightarrow read/write

We take as our starting point the simplest possible machine – a deterministic one-way finite state automaton (**FSA**).

1.1. Finite state automata, and configurations of state machines

The intuition of a finite state automaton is that we have a set of states Q , together with a set Σ of unary (possibly partial[†]) functions that act on it. Σ is referred to as the *input*

[†] The fact that we allow partial functions in the definition of determinism is an important theme of this paper. In particular, some of the categorical processes are not defined on categories of global functions. In terms of a computational interpretation, this corresponds to allowing for the possibility of computations that do not terminate, which is required for a discussion of two-way machines.

alphabet. The transitions we wish to model are multiple partial function applications, such as $hgf(a) = hg(b) = h(a) = c$. We draw this computational process as

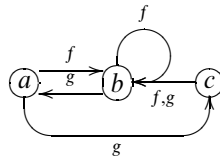
$$hgf^{(a)} \mapsto hg^{(b)}f \mapsto h^{(a)}gf \mapsto {}^{(c)}hgf,$$

both for consistency with the usual concept of a machine head moving over a tape of symbols, and because we do not wish to forget the function we have just applied; in generalisations of this simple machine, ‘forgetting information’ is an explicit computational step, distinct from the process of re-writing the symbol that has just been read.

Definition 1. The steps in the above computational process, which describe the computation of a finite state machine on a tape, are *instantaneous configurations* of the state machine. We define an instantaneous configuration of a machine \mathcal{L} on a tape of length n to be a member of $\Sigma^i \times Q \times \Sigma^j$, where $i + j = n$.

All the algebraic models we introduce are based on the concept of a machine mapping one instantaneous configuration to another. In particular, for our algebraic models, we will be interested in the mappings between configurations of the form $\Sigma^i \times Q \times \Sigma^j$, where either i or j is zero – that is, computations starting and finishing at the ends of the tape.

For finite state automata, we follow the common convention that describes the action of the input alphabet on the set of states by a single ‘next-state (partial) function’, $\circ : \Sigma \times Q \rightarrow Q$. This has a nice graphical interpretation in terms of a ‘transition diagram’, such as



and this gives the alternative intuition of computations of a finite state automaton as ‘following paths through a labelled graph’.

Definition 2. To formalise the above intuition, we define a (deterministic) finite state automaton to be specified by a *state set* Q , an *input alphabet* Σ , and a *next state*, or *transition* (partial) function $\circ : \Sigma \times Q \rightarrow Q$.

The non-deterministic case is similar enough not to need describing separately; it is enough to note that the set Σ becomes a set of relations on the state set Q , or alternatively, the ‘next state function’ \circ is a relation rather than a function.

1.2. Generalising finite state automata to two-way automata

Definition 3. We can generalise the above definition of a (deterministic) finite state automaton to give the state machine known as a (deterministic) two-way automaton; this comprises a state set Q consisting of two distinct subsets Q_l, Q_r called left-moving and

right-moving states, together with an input alphabet Σ , and a next-state partial function

$$\circ : \Sigma \times Q_l \sqcup Q_r \times \Sigma \rightarrow Q_l \sqcup Q_r.$$

The intended interpretation of the next-state function is as follows:

Consider the following instantaneous configuration of the two-way automaton

$$x_1 \dots x_{i-2} x_{i-1} \stackrel{(a)}{ } x_i x_{i+1} \dots x_n,$$

and assume that a is a left-moving state. Then the two-way automaton can move to either the configuration

$$x_1 \dots x_{i-2} \stackrel{(b)}{ } x_{i-1} x_i \dots x_n \quad b = x_{i-1} \circ a,$$

where b is also a left-moving state, or

$$x_1 \dots x_{i-2} x_{i-1} \stackrel{(c)}{ } x_i x_{i+1} \dots x_n \quad c = x_{i-1} \circ a,$$

where c is a right-moving state.

Alternatively, when a is a right-moving state, the two-way automaton, in the configuration

$$x_1 \dots x_{i-2} x_{i-1} \stackrel{(a)}{ } x_i x_{i+1} \dots x_n$$

can move either to the configuration

$$x_1 \dots x_{i-1} x_i \stackrel{(d)}{ } x_{i+1} \dots x_n \quad d = a \circ x_i,$$

when d is also a right-moving state, or to the configuration

$$x_1 \dots x_{i-2} x_{i-1} \stackrel{(e)}{ } x_i x_{i+1} \dots x_n \quad e = a \circ x_i,$$

where e is a left-moving state.

Note that this definition is very similar, but not identical, to the definition of a two-way automaton given by J.-C. Birget in Birget (1989a). However, the distinction is subtle, so we postpone discussion of the differences to Appendix A.

The extension of our definition to the non-deterministic case is relatively straightforward. However, it should be noted that there are two qualitatively different ways in which non-determinism can arise; the next-state function can be replaced by a next-state relation, and we can also drop the requirement that the set of left-moving states and the set of right-moving states are disjoint (this is equivalent to introducing a partial bijection between Q_l and Q_r). When we refer to a non-deterministic two-way automaton, we allow for either of these possibilities; we assume that non-determinism means that we have a ‘next possible set of configurations’, rather than a number of distinct labels on the read-head of the machine.

1.3. Generalising finite state automata to read/write machines

One-way automata with write capability are referred to as *Mealy machines*, or *Moore machines*, according to whether the output is associated with the transitions, or with the states (an analogous situation to the question of whether the read-head of a tape based

state machine points at or between cells on the tape). As the two definitions are equivalent, we restrict our attention to Mealy machines, for consistency with the other state machines we present, and in accordance with our original intuition of finite state automata as a description of applying functions to state sets.

Definition 4. A (deterministic) Mealy machine (a one-way automaton with output) is specified by the following data: a set of states Q , an input alphabet Σ , and a transition/output partial function

$$\circ : \Sigma \times Q \rightarrow Q \times \Sigma.$$

The computational interpretation of the transition/output function is that we have a read/write head labelled by a state, moving from right to left over a string of symbols written on a tape. When the read/write head passes over a symbol on the tape, the label on the read/write head changes, and the symbol written on the tape (on the cell that has just been read) also changes, as specified by the transition/output function.

Diagrammatically, if we have an instantaneous configuration of the Mealy machine

$$x_1 \dots x_{i-2} x_{i-1} \stackrel{(a)}{x_i x_{i+1} \dots x_n},$$

the next configuration is

$$x_1 \dots x_{i-2} \stackrel{(b)}{y} x_i \dots x_n \quad (b, y) = x_{i-1} \circ a.$$

The extension to the non-deterministic case is again trivial; we merely replace the transition/output partial function by a transition/output relation. However, we again assume that non-determinism means the existence of ‘a set of next possible configurations’, so we do not treat non-determinism in the state, and non-determinism in the output symbol, differently.

1.4. Generalising to two-way read/write machines

Our original intuition of a finite state automaton was as a read-head moving over a tape, changing state as it goes. We wish to generalise this to a machine that can move in both directions and write on the tape as well as changing state. This gives a machine that is very similar to a bounded Turing machine[†]. The difference is that, in our conception, the machine head points between adjacent cells on a tape (or between a cell and the end of the tape), whereas in the original description of a Turing machine (as found in Cohen (1991), for example) – bounded or otherwise – the machine head points at a single cell on the tape. We require our definition in order to present simpler state machines as specific examples of more complex state machines without *ad hoc* constructions (such as those required for the demonstration of the equivalence of two-way automata as described in Shepherdson (1959), and two-way automata as described in Birget (1989a)).

[†] We distinguish between a *Bounded Turing machine* where the machine head is constrained by the length of the input tape, and a *Linear Bounded Turing machine* where the machine head is constrained by a linear function of the length of the input tape.

Our description of a bounded Turing machine is given by ‘merging the definition of a Mealy machine with that of a two-way automaton’. This gives a set of states $Q = Q_l \cup Q_r$ satisfying $Q_l \cap Q_r = \emptyset$, an alphabet Σ , and a transition/rewrite partial function

$$\circ : \Sigma \times Q_l \sqcup Q_r \times \Sigma \rightarrow Q_l \times \Sigma \sqcup \Sigma \times Q_r.$$

The computational interpretation of this function is as follows: consider an instantaneous configuration of the tape

$$x_1 \dots x_{i-2} x_{i-1} \stackrel{(a)}{x_i} x_{i+1} \dots x_n,$$

and let a be a left-moving state. Then the bounded Turing machine can move to either the configuration

$$x_1 \dots x_{i-2} \stackrel{(b)}{y} x_i \dots x_n \quad (b, y) = x_{i-1} \circ a,$$

where b is also a left-moving state, or

$$x_1 \dots x_{i-2} z \stackrel{(c)}{x_i} x_{i+1} \dots x_n \quad (z, c) = x_{i-1} \circ a,$$

where c is a right-moving state.

Dually, when a is a right-moving state, the instantaneous configuration

$$x_1 \dots x_{i-2} x_{i-1} \stackrel{(a)}{x_i} x_{i+1} \dots x_n$$

can move to either the configuration

$$x_1 \dots x_{i-1} y \stackrel{(d)}{x_{i+1}} \dots x_n \quad (d, y) = a \circ x_i$$

when d is also a right-moving state, or

$$x_1 \dots x_{i-2} z \stackrel{(e)}{x_i} x_{i+1} \dots x_n \quad (z, e) = a \circ x_i$$

where e is a left-moving state.

Although there is no standard definition of a (bounded) Turing machine where the machine head is positioned between cells of the tape with which we can compare our definition, there is still a subtle distinction with how a bounded Turing machine is usually described. This is also covered in Appendix A, as a simple extension of the same considerations for two-way automata.

Finally, the extension to the non-deterministic case is not problematic, given the proviso that non-determinism is thought of as ‘having a set of next possible configurations’.

2. Summary of state machines considered

By starting with the definition of a deterministic (one-way) finite state automaton, and considering how to generalise it by allowing non-determinism, two-way action and write capability, we have described both the deterministic and non-deterministic version of the following machines:

- Finite State Automata, **FSA**
- Automata with output, Mealy Machines, **MM**
- Two-way Finite State Automata, **2-FSA**
- (Strictly) Bounded Turing Machines, **BTM**

By representing the three distinct generalisation processes as orthogonal axes, we obtain the following diagram:

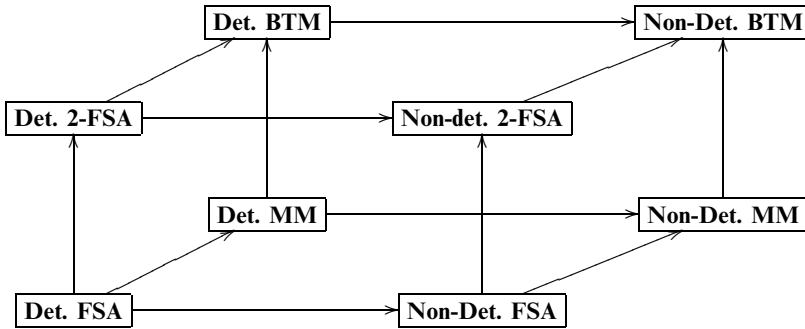


Fig. 1. Generalisations of Finite State Automata

We give algebraic models for each of these machines, and give a categorical interpretation of each of the generalisation processes. Each arrow in the above diagram corresponds to embedding a simpler algebraic model into more complex algebraic model, as a special case. The algebraic/categorical embeddings described correspond to considering a read-only machine as a special case of a read/write machine, a one-way machine as a special case of a two-way machine, or a deterministic machine as a special case of a non-deterministic machine.

The simplest generalisation, the embedding of models of deterministic machines into models of non-deterministic machines, is given by the standard embedding of the category of partial functions into the category of relations in each case, so we briefly describe this embedding in each case, but restrict our comments to observations on the embedding, and computational interpretations.

Conventions

We have adopted a number of conventions in our descriptions of the above machines. In order to clarify the terms of reference, these are as follows:

- (i) **The definition of Deterministic allows for the assumption of partiality.** In particular, the definition of two-way automata and bounded Turing machines in the deterministic case requires the acceptance of partiality (as the algebraic representation of non-terminating computations) in order for the algebraic models to be well-defined.
- (ii) **The definition of non-determinism should be thought of as having ‘a number of next configurations’, for each of the machines listed.** This is in opposition to having a specific definition for each machine, such as (for example) a read head labelled by a number of distinct states, a cell on a tape labelled by a number of distinct symbols, or a machine head on a two-way machine that splits into both a left-moving and a right-moving part. This allows us to treat non-determinism in a consistent way for all the state machines described.

- (iii) Both two-way automata and bounded Turing Machines are described in the same way as one-way finite state automata. **The ‘read head’, or ‘pointer’ is positioned between cells on a tape, rather than pointing at a single cell.** This is for consistency with the other finite state machines, and is required for the construction of algebraic models.
- (iv) **Bounded Turing Machines are strictly bounded by the length of the input word.** This is required for a consistent description of all the state machines we consider, and may be justified by the well-known conversion of linear bounded Turing machines into strictly bounded Turing machines by a modification of the state set.

3. An overview of the categorical constructions used

We take as our starting point the category of partial maps on sets, which we denote **pSet**. In Section 6, we give the classical construction of how transitions of deterministic finite state machines are modelled by monoid homomorphisms from the free monoid on an ‘input alphabet’ to (finite) endomorphism monoids of **pSet**; these are referred to as the *transition homomorphisms*, in, for example, Cohen (1991). We refer to **pSet** as the *base category*, and construct the ‘left-hand side’ of the above cube in terms of categorical constructions applied to it. The ‘right-hand side’ of the above cube is constructed in the same way, but using the category of relations on sets, which we denote **rSet**, as the base category.

This then leaves two categorical constructions to account for: they are needed to model the generalisation from one-way to two-way machines, and from read-only machines to read/write machines.

The categorical representation of the one-way to two-way generalisation is given by a construction of Joyal, Street and Verity, referred to as the **Int** construction, which gives a canonical construction of a compact closed category from a traced symmetric monoidal category (Joyal *et al.* 1996). This is equivalent to the ‘Geometry of Interaction’ construction, an important tool in modelling the process of cut-elimination in linear logic, as given in Abramsky (1996).

In Section 7 we will introduce a construction giving the categorical representation of the read-only to read/write generalisation. We refer to this construction as the **Comp** construction, because of its use in modelling the entire computation of a finite state machine as a single homomorphism. This construction is a method of constructing *graded categories* from arbitrary monoidal categories. For the algebraic models we consider, the main interest in this construction is the (graded) endomorphism monoids.

The 8 finite state machines derived from Figure 1 are as follows:

- 1 deterministic one-way read-only automata (possibly partial)
- 2 non-deterministic one-way read-only automata
- 3 deterministic two-way read-only automata (must be partial)
- 4 non-deterministic two-way read-only automata
- 5 deterministic one-way read/write automata (possibly partial)
- 6 non-deterministic one-way read/write automata
- 7 deterministic two-way read/write automata (must be partial)
- 8 non-deterministic two-way read/write automata.

We adopt the convention of denoting the input/output alphabet by Σ , and the set of states by Q , or $Q = Q_l \cup Q_r$ in the two-way case.

Our algebraic models are then constructed as monoid homomorphisms in each case; we denote the endomorphism monoid of an object Q by $End(Q)$. The above finite state machines then have their transitions modelled by the following algebraic structures:

- 1 monoid hom.: $\Sigma^* \rightarrow End(Q)$, where $Q \in Ob(\mathbf{pSet})$
- 2 monoid hom.: $\Sigma^* \rightarrow End(Q)$, where $Q \in Ob(\mathbf{rSet})$
- 3 monoid hom.: $\Sigma^* \rightarrow End(Q_l, Q_r)$, where $(Q_l, Q_r) \in Ob(\mathbf{Int}(\mathbf{pSet}))$
- 4 monoid hom.: $\Sigma^* \rightarrow End(Q_l, Q_r)$, where $(Q_l, Q_r) \in Ob(\mathbf{Int}(\mathbf{rSet}))$
- 5 graded monoid hom. $\mathbb{N} \rightarrow End(Q)$, where $Q \in \mathbf{Comp}_\Sigma(\mathbf{pSet})$
- 6 graded monoid hom $\mathbb{N} \rightarrow End(Q)$, where $Q \in \mathbf{Comp}_\Sigma(\mathbf{rSet})$
- 7 graded monoid hom. $\mathbb{N} \rightarrow End(Q_l, Q_r)$, where $(Q_l, Q_r) \in \mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{pSet}))$
- 8 graded monoid hom. $\mathbb{N} \rightarrow End(Q_l, Q_r)$, where $(Q_l, Q_r) \in \mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{rSet}))$.

In terms of algebraic models (as opposed to state machines), the non-deterministic case is better known than the deterministic case, so we first give the categorical constructions based on the *Category of Relations*, \mathbf{rSet} , and then demonstrate how the deterministic case can be modelled by constructions based on the *Category of Partial Functions*, \mathbf{pSet} , as a special case. However, for the other two generalisation processes, we start at the bottom left of the cube. We also demonstrate how the representation of a one-way machine as a special case of a two-way machine, and the representation of a read-only machine as a special case of a read/write machine correspond to canonical embeddings of algebraic structures.

We now give the general definitions and constructions of the categories and categorical processes we require.

4. The Int construction

The categorical framework we demonstrate to be required for modelling the transitions of two-way finite state machines is found in the theory of compact closed and traced monoidal categories. The following definition is equivalent to that of Joyal *et al.* (1996).

Definition 5.

- (i) A *traced symmetric monoidal category* is a symmetric monoidal category[†] $(\mathbf{C}, \otimes, s_{X,Y})$ equipped with a family of functions

$$Tr_{A,B}^U : \mathbf{C}(X \otimes U, Y \otimes U) \rightarrow \mathbf{C}(X, Y)$$

that are natural in X, Y , and satisfy:

- (a) Given $f : X \otimes I \rightarrow Y \otimes I$, we have

$$Tr_{X,Y}^I(f) = \rho f \rho^{-1} : X \rightarrow Y$$

[†] For our purposes, we assume that the canonical associativity isomorphisms for the monoidal structure are strict, and refer to Joyal *et al.* (1996) for the full definition.

(b) Given $f : X \otimes U \rightarrow Y \otimes U'$ and $g : U' \rightarrow U$, we have

$$Tr_{X,Y}^U((1_Y \otimes g)f) = Tr_{X,Y}^{U'}(f(1_X \otimes g))$$

(c) Given $f : X \otimes U \otimes V \rightarrow Y \otimes U \otimes V$, we have

$$Tr_{X,Y}^{U \otimes V}(f) = Tr_{X,Y}^U(Tr_{X \otimes U, Y \otimes U}(f)).$$

(d) Given $f : X \otimes U \rightarrow Y \otimes U$, and $g : W \rightarrow Z$, we have

$$g \otimes Tr_{X,Y}^U(f) = Tr_{W \otimes X, Z \otimes Y}^U(g \otimes f)$$

(e) $Tr_{U,U}^U(s_{U,U}) = 1_U$.

(ii) A *compact closed category* is a symmetric monoidal category[†] $(\mathbf{C}, \otimes, s_{X,Y}, \rho, \lambda)$, together with a map $(\)^\vee$ that takes an object A to its *left dual* A^\vee . Also, for every $A \in Ob(\mathbf{C})$, there exist two morphisms, the counit map $\epsilon_A : A^\vee \otimes A \rightarrow I$ and the unit map $\eta_A : I \rightarrow A \otimes A^\vee$, that satisfy the following coherence conditions:

(a) $\rho_A(1_A \otimes \epsilon_A)(\eta_A \otimes 1_A)\lambda_A^{-1} = 1_A$

(b) $\lambda_{A^\vee}(\epsilon_A \otimes 1_{A^\vee})(1_{A^\vee} \otimes \eta_A)\rho_{A^\vee}^{-1} = 1_{A^\vee}$

In Joyal *et al.* (1996), it is proved that every compact closed category also has a trace defined on it, called the *canonical trace*.

Theorem 1. In any compact closed category $(\mathbf{C}, \otimes, \lambda, \rho, \epsilon, \eta, (\)^\vee)$ a trace is defined by

$$Tr_{A,B}^U(f) = \rho_B(1_B \otimes \epsilon'_{U^\vee})(f \otimes 1_{U^\vee})(1_A \otimes \eta_U)\rho_A^{-1}.$$

The same paper (Joyal *et al.* 1996) introduced a categorical construction of a compact closed category from a traced monoidal category[‡]. This construction is equivalent to the ‘Geometry of Interaction’ construction, as found in Abramsky (1996). This has significant applications to the field of linear logic, in particular the representation of linear logic given by the Geometry of Interaction system (Girard 1988a; Girard 1988b).

Theorem 2. Let \mathbf{V} be a symmetric traced monoidal category. A compact closed category, denoted $\mathbf{Int}(\mathbf{V})$, can be defined in terms of the objects and arrows of \mathbf{V} , and there exists an embedding of \mathbf{V} into $\mathbf{Int}(\mathbf{V})$, where the trace on \mathbf{V} is the canonical trace of $\mathbf{Int}(\mathbf{V})$.

This result is the symmetric case of the main theorem of Joyal *et al.* (1996). We do not reproduce this proof, but give the construction of the category $\mathbf{Int}(\mathbf{V})$ in terms of the objects, morphisms, and traced monoidal structure of \mathbf{V} .

Objects: The objects of $\mathbf{Int}(\mathbf{V})$ are defined to be pairs of objects of \mathbf{V} , so

$$X, U \in Ob(\mathbf{V}) \Leftrightarrow (X, U) \in Ob(\mathbf{Int}(\mathbf{V})).$$

[†] Again, we give the case where the associativity is strict, and refer to Joyal *et al.* (1996) for the full definition.

[‡] In fact, this was the construction of tortile monoidal categories from arbitrary traced monoidal categories. However, for our discussion of finite state machines, we only require the symmetric monoidal traced/compact closed case.

The unit object is given by (I, I) , where I is the unit object for the monoidal structure of \mathbf{V} .

Arrows: The arrows of $\mathbf{Int}(\mathbf{V})$ are defined in terms of arrows of \mathbf{V} . An arrow $F: (X, U) \rightarrow (Y, V)$ in $\mathbf{Int}(\mathbf{V})$ is specified by an arrow $f: X \otimes V \rightarrow Y \otimes U$, so there is a bijection of arrows between $\mathbf{V}(X \otimes V, Y \otimes U)$ and $\mathbf{Int}(\mathbf{V})((X, U), (Y, V))$ for all objects X, Y, U, V of \mathbf{V} .

Composition of Arrows: The composition of arrows in $\mathbf{Int}(\mathbf{V})$ is defined in terms of the composition of \mathbf{V} , and the categorical trace:

Given $F \in \mathbf{Int}(\mathbf{V})((X, U), (Y, V))$, and $G \in \mathbf{Int}(\mathbf{V})((Y, V), (Z, W))$, specified by arrows $f \in \mathbf{V}(X \otimes V, Y \otimes U)$, and $g \in \mathbf{V}(Y \otimes W, Z \otimes V)$, respectively, we specify the composite $GF \in \mathbf{Int}(\mathbf{V})((X, U), (Z, W))$ to be

$$Tr_{X \otimes W, Z \otimes U}^V((1_Z \otimes s_{VU})(g \otimes 1_U)(1_Y \otimes s_{UW})(f \otimes 1_W)(1_X \otimes s_{W,V})).$$

The motivation for this definition, as ‘each object has both a forward-moving and a backward-moving part’, can be seen diagrammatically either in Joyal *et al.* (1996), or in the special case of the category of relations in Subsection 4.1.

Identities: For all objects (X, U) , the identity at the object (X, U) is specified by $1_X \otimes 1_U$.

The monoidal tensor: The monoidal tensor $\otimes: \mathbf{Int}(\mathbf{V}) \times \mathbf{Int}(\mathbf{V}) \rightarrow \mathbf{Int}(\mathbf{V})$ is defined in terms of the monoidal tensor of \mathbf{V} (by convention, we use the same symbol for both). On objects, it is defined by $(X, U) \otimes (X', U') = (X \otimes X', U' \otimes U)$, and on arrows, by, for all $F: (X, U) \rightarrow (Y, V)$, and $F': (X', U') \rightarrow (Y', V')$,

$$(F \otimes F'): (X \otimes X', U' \otimes U) \rightarrow (Y \otimes Y', V' \otimes V)$$

is specified by

$$((1 \otimes f') \otimes 1)(s \otimes s)((1 \otimes f) \otimes 1)(s \otimes s).$$

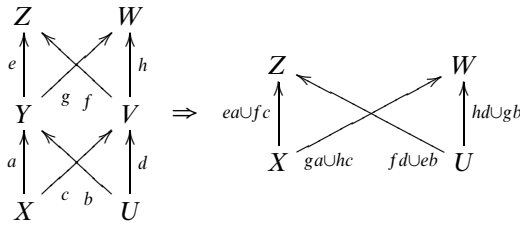
Where $F, F' \in \mathbf{Int}(\mathbf{V})$ are specified by $f, f' \in \mathbf{V}$, respectively. (Note that the subscripts on the canonical isomorphisms have been omitted for clarity).

The canonical embedding: The canonical embedding of V into $\mathbf{Int}(\mathbf{V})$ is given on objects in terms of the unit object by $X \mapsto (X, I)$, and on arrows by $f \mapsto f \otimes 1_I$. It is straightforward to check that this embedding preserves both the tensor and the categorical trace (or, more precisely, maps the categorical trace of \mathbf{V} to the canonical trace of $\mathbf{Int}(\mathbf{V})$).

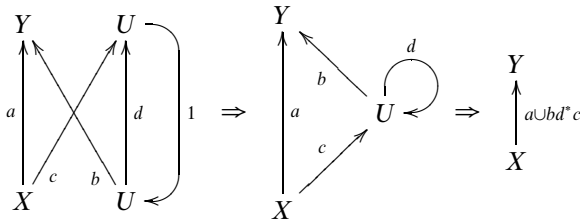
4.1. The trace on \mathbf{rSet} , and the category $\mathbf{Int}(\mathbf{rSet})$

In the presentation of the \mathbf{Int} construction given in Joyal *et al.* (1996), the example of the category of relations on sets is given as a special case. The category \mathbf{rSet} has all sets as objects and relations between sets as arrows (so that the endomorphism monoid of an object X is the monoid $B(X)$ of binary relations on X). \mathbf{rSet} also has a monoidal tensor, given by $X \sqcup Y = X \times \{0\} \cup Y \times \{1\}$, together with the natural extension to arrows. An arrow $R: X \sqcup U \rightarrow Y \sqcup V$ then has a natural matrix representation as $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ where $a \in \mathbf{rSet}(X, Y)$, $b \in \mathbf{rSet}(U, Y)$, $c \in \mathbf{rSet}(X, V)$, $d \in \mathbf{rSet}(U, V)$. Composition of relations in

this form is given by the usual definition of matrix multiplication, where multiplication is interpreted by composition of relations, and addition is interpreted by set-theoretic union. Diagrammatically, this can be drawn as follows:



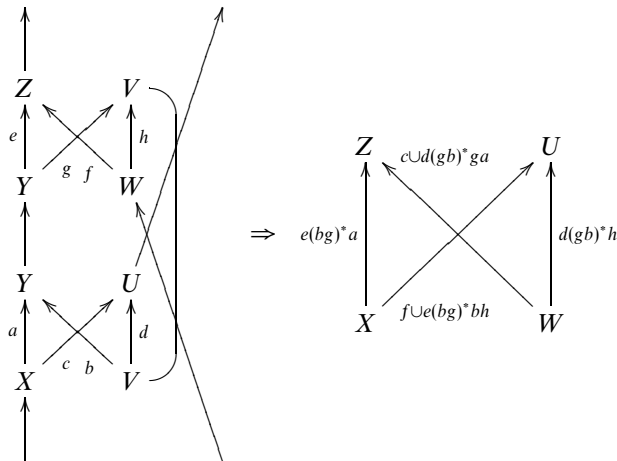
As demonstrated in Joyal *et al.* (1996), the category \mathbf{rSet}, \sqcup has a trace. Using the same diagrammatic notation as for the composition of matrices of relations, the trace can be represented as the introduction of a feedback loop, as follows:



Note the use of the *Kleene star*, or *reflexive transitive closure* of the relation d , defined by $d^* = 1 \cup d \cup d^2 \cup d^3 \cup \dots$. In matrix notation, we write

$$Tr \begin{pmatrix} a & b \\ c & d \end{pmatrix} = a \cup bd^* c.$$

As \mathbf{rSet} has a categorical trace, by Joyal *et al.* (1996), it can be used to construct a compact closed category $\mathbf{Int}(\mathbf{rSet})$. This category has pairs of objects of \mathbf{rSet} as objects, and arrows are specified by matrices of relations as above. However, the composition is defined diagrammatically:



Intuitively, this is ‘using the categorical trace to reverse the order of composition in the second variable’. Using the matrix notation, we write

$$\begin{pmatrix} e & f \\ g & h \end{pmatrix} \cdot \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} e(bg)^*a & f \cup e(bg)^*bh \\ c \cup d(gb)^*ga & d(gb)^*h \end{pmatrix}.$$

(We denote the composition by $M \cdot N$, in order to distinguish it from the standard composition of matrices of relations).

5. The Comp construction

The following construction gives the categorical framework for modelling finite state machines with read/write capability. This construction is a method of taking a category \mathbf{C} with a set action to a *graded category*, $\mathbf{Comp}(\mathbf{C})$. A graded category is considered to be a simple extension of the usual definition of a graded monoid to the many-object case.

Definition 6.

- (i) A *graded monoid* is the data specified by a monoid homomorphism $g : M \rightarrow (\mathbb{N}, +)$, or, equivalently, a sequence of sets M_0, M_1, M_2, \dots together with an associative operation $\odot : M_i \times M_j \rightarrow M_{i+j}$, and an identity for this operation, $1 \in M_0$.
- (ii) A *Graded Category* \mathbf{D} is a simple generalisation of the above definition, so a graded category is the data specified by a functor from a category \mathbf{C} to the monoid $(\mathbb{N}, +)$, considered as a one-object category.

Equivalently, a graded category \mathbf{D} consists of a class of objects $Ob(\mathbf{D})$, where for every pair of objects X, Y , and every $n \in \mathbb{N}$, there exists a set of morphisms $\mathbf{D}_n(X, Y)$, together with an associative composition $\odot : \mathbf{D}_m(Y, Z) \times \mathbf{D}_n(X, Y) \rightarrow \mathbf{D}_{n+m}(X, Z)$. There also exist identity morphisms at each object, $1_X \in \mathbf{D}_0(X, X)$, satisfying the usual conditions for identity arrows, namely $1_X f = f$, and $g 1_X = g$ for appropriate f and g .

For each object X , the set $\{\mathbf{D}_i(X, X) : i \in \mathbb{N}\}$, together with the graded composition \odot , specifies a graded monoid. We denote this by $End(X) \in \mathbf{D}$, with the proviso that $End(X)$ refers to a *graded monoid*.

Our construction is then as follows.

Definition 7. Let \mathbf{C}, \otimes be a monoidal category. We assume strict associativity and strict unit arrows, so the iterated tensor $\Sigma^n = \Sigma \otimes \Sigma \otimes \dots \otimes \Sigma$ has a unique meaning for any $\Sigma \in Ob(\mathbf{C})$, and $n \in \mathbb{N}$.

For any $\Sigma \in Ob(\mathbf{C})$, we define $\mathbf{Comp}_\Sigma(\mathbf{C})$ to have the same objects as \mathbf{C} , and for any pair of objects X, Y , and integer $n \in \mathbb{N}$, we define $\mathbf{Comp}_\Sigma(\mathbf{C})_n(X, Y)$ to be the set $\mathbf{C}(\Sigma^n \otimes X, Y \otimes \Sigma^n)$.

We define a composition \odot on $\mathbf{Comp}_\Sigma(\mathbf{C})$ as follows:

Given $f \in \mathbf{Comp}_\Sigma(\mathbf{C})_i(X, Y)$ and $g \in \mathbf{Comp}_\Sigma(\mathbf{C})_j(Y, Z)$, we define

$$g \circ f = (g \otimes 1_{\Sigma^i})(1_{\Sigma^j} \otimes f),$$

where this composition is taken in \mathbf{C} .

Theorem 3. The *Comp* construction, as defined above, specifies a graded category.

Proof. For all $f \in \mathbf{Comp}_{\Sigma}(\mathbf{C})_i(X, Y)$ and $g \in \mathbf{Comp}_{\Sigma}(\mathbf{C})_j(Y, Z)$, this composition exists since (in the category \mathbf{C})

$$f \in \mathbf{C}(\Sigma^i \otimes X, Y \otimes \Sigma^i), \quad g \in \mathbf{C}(\Sigma^j \otimes Y, Z \otimes \Sigma^j),$$

so

$$(1_{\Sigma^j} \otimes f) \in \mathbf{C}(\Sigma^{i+j} \otimes X, \Sigma^j \otimes Y \otimes \Sigma^i), \quad (g \otimes 1_{\Sigma^i}) \in \mathbf{C}(\Sigma^j \otimes Y \otimes \Sigma^i, Z \otimes \Sigma^{i+j}).$$

Therefore, $g \circ f$ is well-defined in \mathbf{C} , and a member of $\mathbf{C}(\Sigma^{i+j} \otimes X, Z \otimes \Sigma^{i+j})$, as required. Associativity of \circ follows immediately from the associativity of composition in \mathbf{C} and the assumption of strict associativity for the monoidal tensor \otimes .

Finally, the identities are specified by $1_X \in \mathbf{C}(\Sigma^0 \otimes X, X \otimes \Sigma^0)$ for all objects X , which is consistent with the graded category composition, since the unit arrows are assumed to be strict, and thus we may make the definition $\Sigma^0 \otimes X = I \otimes X = X$. \square

This completes our summary of the algebraic structures required. In the following sections we demonstrate how these algebraic structures model transitions of the finite state machines presented in Section 2.

6. Finite state automata, and the categories of relations and partial functions

In the remainder of this paper, we consider structures built on the categories of partial functions on sets and relations on sets. We denote these by \mathbf{pSet} and \mathbf{rSet} , respectively. Unless explicitly stated otherwise, we will consider constructions on *finite* sets – the extension of the constructions considered to infinite sets does not appear to be problematic, but is not helpful for our discussion of finite state machines.

We start with the standard construction of the transition homomorphism of a finite state automaton. Although this is well known, and may be found in any undergraduate-level text, we give the basic definitions and construction, in order to specify the formalism and notation that we use. We have seen in Subsection 1.1 that a finite state automaton \mathcal{A} is specified by the following data:

An input alphabet Σ , a state set Q , and a next state function $\circ \in \mathbf{pSet}(\Sigma \times Q, Q)$, for a deterministic machine, or a next state relation $\circ \in \mathbf{rSet}(\Sigma \times Q, Q)$ for a non-deterministic machine.

The usual model of the transitions of a finite state automaton, as found in, for example, Howie (1991), is in terms of a monoid homomorphism, called the *transition homomorphism*.

Definition 8. Given a finite state automaton \mathcal{A} , we may curry the next state function (respectively, relation) to construct a transition function $t : \Sigma \rightarrow \mathbf{pSet}(Q, Q)$ in the deterministic case, or $t : \Sigma \rightarrow \mathbf{rSet}(Q, Q)$ in the non-deterministic case. We use $\mathit{End}(Q)$ to denote the endomorphism monoid of Q in both cases.

We then consider the free monoid Σ^* , consisting of all words over Σ , including the empty word λ . As Σ^* is a free monoid on $n = |\Sigma|$ generators, the function $t : \Sigma \rightarrow \mathit{End}(Q)$

can be lifted to a unique monoid homomorphism $\bar{t} : \Sigma^* \rightarrow \text{End}(Q)$, known as the *transition homomorphism*.

Note that the congruence \sim_t induced on Σ^* by \bar{t} satisfies Σ^* / \sim_t is isomorphic to the submonoid of $\text{End}(Q)$ generated by $\{t(x) : x \in \Sigma\}$. Hence, Σ^* / \sim_t is known as the *transition monoid*.

The computational interpretation of the transition homomorphism is the following: when we have a word of symbols from Σ written on a tape, and a *pointer*, or *read head*, labelled by a state symbol moving from right to left over the tape, changing state symbols as it goes, the function t describes all computations on a tape with one symbol. When we have more than one symbol on the tape, the concatenation of state changes is given by composition in the category of partial functions, or relations. This lifting process is the algebraic reflection of the fact that we consider concatenation of symbols on a tape to be a 'free' associative operation, and we consider the empty tape to have trivial action on the state set.

6.1. Recognition of formal languages by state machines

The motivation for the study of finite state machines in general, and finite state automata in particular, is often the study of the classes of languages recognised by them. In terms of finite state automata, the informal description of the set of words recognised by an automaton A is as follows:

Two specified subsets[†] of the state set are chosen, referred to as the *initial* and *terminal* states, respectively. By convention, these are denoted I and T . A word $w \in \Sigma^*$ is said to be *accepted* by the automaton A iff there exists a labelled path in the automata diagram (the graphical representation of the next-state function) from an initial state $i \in I$ to a terminal state $t \in T$.

In terms of the algebraic representation of the transitions given above, we require the notion of a partial identity 1_S at a subset $S \subseteq Q$, in the monoid $B(Q)$, and say that the automaton \mathcal{A} recognises the word $w \in \Sigma^*$ iff

$$1_T t(w) 1_I \neq 0$$

where 0 is defined to be the partial identity on the empty subset.

As we develop algebraic models of the transitions of all the finite state machines we consider, we take this as the general definition of the recognition of a word by a state machine, and make no further (explicit) reference to the recognition of words and formal languages. Note, however, that adopting this definition already forces us to consider partially defined identities in the algebraic models of deterministic machines.

[†] An alternative, but equivalent, description is to assume the set of initial states has exactly one member, and refer to this as *the initial state*. However, the class of languages recognised is the same in both cases, so we adopt the more symmetric definition.

7. Two-way automata and the Int construction

Recall the definition of two-way automata, and the description of their action given in Definition 3.

The usual algebraic models of transitions of two-way automata are due to J.-C. Birget. In Birget (1989a), he constructs algebraic models of (non-deterministic) two-way automata, as follows.

Definition 9. For every word of the input alphabet w , the *global transition relations* of w are four arrows in \mathbf{rSet} defined in terms of the action of the two-way automaton:

- $[\Leftarrow w] \in \mathbf{rSet}(Q_r, Q_l)$, consisting of the set of pairs (q', q) for which there exists a computation starting in configuration ${}^{(q)}w$ and finishing in configuration ${}^{(q')}w$.
- $[-w \rightarrow] \in \mathbf{rSet}(Q_r, Q_r)$, consisting of the set of pairs (q', q) for which there exists a computation starting in configuration ${}^{(q)}w$ and finishing in configuration $w^{(q')}$.
- $[\leftarrow w-] \in \mathbf{rSet}(Q_l, Q_l)$, consisting of the set of pairs (q', q) for which there exists a computation starting in configuration $w^{(q)}$ and finishing in configuration ${}^{(q')}w$.
- $[w \Rightarrow] \in \mathbf{rSet}(Q_l, Q_r)$, consisting of the set of pairs (q', q) for which there exists a computation starting in configuration $w^{(q)}$ and finishing in configuration $w^{(q')}$.

These relations also feature implicitly[‡] in the earlier work Shepherdson (1959).

It is natural to write these four relations in matrix form, and we adopt the following convention:

$$[w] = \begin{pmatrix} [\leftarrow w-] & [\Leftarrow w] \\ [w \Rightarrow] & [-w \rightarrow] \end{pmatrix}$$

Note that $[w]$ specifies a single relation in $\mathbf{rSet}(Q_l \sqcup Q_r, Q_r \sqcup Q_l)$, and so is a member of the endomorphism monoid of the object (Q_l, Q_r) in $\mathbf{Int}(\mathbf{rSet})$.

These 4-tuples of relations have an algebraic description for their composition, in that for any two words $u, v \in \Sigma^*$, the global transition relations of the composite $uv \in \Sigma^*$ can be written in terms of their global transition relations, and operations in the category of relations.

In Birget (1989b), it is proved that the global transition relations of the word uv can be expressed in terms of the global transition relations of the words u and v , as follows.

Theorem 4. Given a (non-deterministic) two-way automaton $\mathcal{A} = (Q_l, Q_r, \Sigma, \circ)$, and $u, v \in \Sigma^*$, the global transition relations of the composite uv can be written in terms of the global transition relations for u and v , and operations in the category of relations, as follows:

- $[-uv \rightarrow] = [-v \rightarrow]([u \Rightarrow][\Leftarrow v])^*[-u \rightarrow]$,
- $[uv \Rightarrow] = [v \Rightarrow] \cup [-v \rightarrow][u \Rightarrow]([\Leftarrow v][u \Rightarrow])^*[\leftarrow v-]$,
- $[\Leftarrow uv] = [\Leftarrow u] \cup [\leftarrow u-][\Leftarrow v]([u \Rightarrow][\Leftarrow v])^*[-u \rightarrow]$,
- $[\leftarrow uv-] = [\leftarrow u-]([\Leftarrow v][u \Rightarrow])^*[\leftarrow v-]$.

From this Theorem, the following is immediate from the constructions of Section 4.

[‡] There was a significant difference in that Shepardson considered two-way automata to be defined in a similar way to the usual description of Turing machines, with the ‘Read head’ pointing directly at cells on the tape, rather than between cells.

Theorem 5. The formula for $[vu]$ in terms of $[v]$ and $[u]$ given above is the composition of arrows in the endomorphism monoid of (Q_l, Q_r) in the category $\mathbf{Int}(\mathbf{rSet})$.

Proof. The proof follows immediately by comparing the composition of $\mathbf{Int}(\mathbf{rSet})$ from Subsection 4.1 with Birget’s equations above. \square

Hence, given the (computationally reasonable) assumption that the empty word $\lambda \in \Sigma^*$ satisfies $[\lambda] = 1_{(Q_l, Q_r)} \in \mathbf{Int}(\mathbf{rSet})$, we have described (non-deterministic) two-way automata as monoid homomorphisms from Σ^* to endomorphism monoids in $\mathbf{Int}(\mathbf{rSet})$, as required for Step 4 of the programme laid out in Section 3. So, in the same way as a (non-deterministic) finite state automaton can be described in terms of a monoid homomorphism from Σ^* to the endomorphism monoid of Q in \mathbf{rSet} , a non-deterministic two-way automaton can be described in terms of a monoid homomorphism from Σ^* to the endomorphism monoid of (Q_l, Q_r) in $\mathbf{Int}(\mathbf{rSet})$.

7.1. Two-way automata – the deterministic case

A deterministic two-way automaton is a special type of non-deterministic two-way automaton, in which the next-state relation $\circ \in \mathbf{rSet}(\Sigma \times Q_l \sqcup Q_r \times \Sigma, Q_l \sqcup Q_r)$ is a partial function.

In this case, when we split the next-state function into its components, we have:

- $[\leftarrow x -] \in \mathbf{pSet}(Q_l, Q_l)$
- $[\rightleftharpoons x] \in \mathbf{pSet}(Q_r, Q_l)$
- $[x \rightleftharpoons] \in \mathbf{pSet}(Q_l, Q_r)$
- $[-x \rightarrow] \in \mathbf{pSet}(Q_r, Q_r)$.

and these four partial functions must satisfy:

- $[\leftarrow x -]$ and $[x \rightleftharpoons]$ have disjoint domains
- $[-x \rightarrow]$ and $[\rightleftharpoons x]$ have disjoint domains.

If we write this in matrix form, as in Section 4, this condition becomes the requirement that entries in the same column of the matrix have disjoint domains[†].

It is trivial to check that the restriction of the categorical trace on the category of relations to the category of partial injective maps, considered as a sub-category, is well-defined – if $R \in \mathbf{rSet}(X \sqcup U, Y \sqcup U)$ is a partial function, the trace $Tr(R) \in \mathbf{rSet}(X, Y)$ is also a partial function. Hence we may talk about the compact closed category $\mathbf{Int}(\mathbf{pSet})$, and thus $[x]$ is a member the endomorphism monoid of (Q_l, Q_r) in $\mathbf{Int}(\mathbf{pSet})$, for all $x \in \Sigma$.

This then gives us an algebraic model of deterministic two-way automata as monoid homomorphisms from Σ^* to $\mathbf{Int}(\mathbf{pSet})$ with no additional work. This is Step 3 of the programme laid out in Section 3.

In terms of a computational interpretation, it should be noted that the category of *total* functions on sets together with the tensor given by disjoint union does not have a categorical trace, and thus cannot be used to construct a compact closed category.

[†] Adding the dual condition, that entries in the same row of the matrix have disjoint images, would imply that the function \circ is a (partial) bijective function, which gives the definition of a reversible two-way automata.

Computationally, this should be interpreted as the possibility that the individual steps of a two-way automaton are deterministic, but the computation as a whole does not terminate.

7.2. One-way automata as special cases of two-way automata

In both the deterministic and non-deterministic case, there is a simple algebraic representation of one-way automata as two-way automata. Recall the construction of the category $\mathbf{Int}(\mathbf{C})$ from a traced symmetric monoidal category \mathbf{C} , and the canonical (trace and tensor preserving) embedding of \mathbf{C} into $\mathbf{Int}(\mathbf{C})$.

Hence, given a monoid homomorphism $t : \Sigma^* \rightarrow \mathbf{rSet}(Q, Q)$, we can compose this with the canonical embedding to give a monoid homomorphism

$$T : \Sigma^* \rightarrow \mathbf{Int}(\mathbf{rSet})((Q, I), (Q, I)),$$

and clearly, the homomorphisms t and T both induce the same congruence on Σ^* .

In terms of two-way automata, this gives a two-way automaton where the set of right-moving states is empty[‡], which is a natural interpretation of finite state automata as special cases of two-way automata.

7.3. The automata-theoretic interpretation of the categorical dual

Recall from Section 4 that every object A of a compact closed category \mathbf{C} has a *left dual*, denoted A^\vee . In Kelly and Laplaza (1980), the dual on arrows is also defined, so for all $f \in \mathbf{C}(X, Y)$ there exists $f^\vee \in \mathbf{C}(Y^\vee, X^\vee)$ satisfying certain natural conditions.

In terms of the \mathbf{Int} construction, the dual of an object (A, B) is simply the object (B, A) , and hence for state machines, the interpretation of the categorical dual is simply that of swapping the role of left-moving and right-moving states in the machine definition.

8. Machines with write capability, and the \mathbf{Comp} construction

We now consider how to form algebraic models of machines with write capability. In the one-way case, we claim that the appropriate algebraic model is given by the application of the \mathbf{Comp} construction to the category of relations or partial functions on sets, and in the two-way case, we claim that the appropriate algebraic model is given by the application of the same construction to the compact closed categories $\mathbf{Int}(\mathbf{rSet})$ and $\mathbf{Int}(\mathbf{pSet})$.

[‡] Of course, a two-way automaton where the set of *left-moving* states is empty is also a one-way machine. The reason for selecting the set of right-moving states to be empty (which is forced on us by the algebraic framework used) is that we are using the *left dual* in the compact closed category, rather than the alternative (but equivalent definition) using the right dual, as found in Kelly and Laplaza (1980). This ensures that the transition homomorphism of a finite state automaton is indeed a homomorphism and not an anti-homomorphism – ultimately, in terms of our original intuition, this is required by the fact that we are writing function application on the left.

9. Algebraic models of Mealy machines

Recall the definition of (non-deterministic) one-way automata with output, Mealy machines, from Definition 4. A Mealy machine is specified by the following data: a set of states Q , an input alphabet Σ , and a transition/output relation,

$$\circ \in \mathbf{rSet}(\Sigma \times Q, Q \times \Sigma).$$

We use this transition/output relation as the basis for our algebraic models, as follows.

Definition 10. Given a Mealy machine $\mathcal{M} = (Q, \Sigma, \circ)$, we define, for all $n \in \mathbb{N}$,

$$[n]_{\mathcal{M}} = \{(q'v, uq)\} \in \mathbf{rSet}(\Sigma^n \times Q, Q \times \Sigma^n)$$

where

$$u^{(q)} \Rightarrow (q')v$$

is a valid computation of the Mealy machine \mathcal{M} . We refer to these relations as the *computation relations* for \mathcal{M} .

Note that, by definition, $[1] = \circ$. Also, when we consider the Cartesian structure of the category of relations, for each $n \in \mathbb{N}$ the computation relation $[n]_{\mathcal{M}}$ uniquely specifies a member of the graded monoid $\mathbf{Comp}_{\Sigma}(\mathbf{rSet})(Q, Q)$. (We are assuming the Cartesian product is associative.)

The **Comp** construction can then be used to provide algebraic models for Mealy machines, as follows.

Theorem 6. Let $\mathcal{M} = (Q, \Sigma, \circ)$ be a Mealy machine, and apply the **Comp** construction, as described in Section 7, to the category \mathbf{rSet} , to form $\mathbf{Comp}_{\Sigma}(\mathbf{rSet})$. Then the computation relations defined in Definition 4 satisfy

$$[n]_{\mathcal{M}} \odot [m]_{\mathcal{M}} = [m + n]_{\mathcal{M}},$$

where \odot denotes the graded composition in this category.

Proof. By definition,

$$[n]_{\mathcal{M}} = \{(q'v, uq)\} \in \mathbf{rSet}(\Sigma^n \times Q, Q \times \Sigma^n),$$

where $u^{(q)} \Rightarrow (q')v$ is a valid computation of the Mealy machine \mathcal{M} on a tape of length n , and

$$[m]_{\mathcal{M}} = \{(q''t, zq')\} \in \mathbf{rSet}(\Sigma^m \times Q, Q \times \Sigma^m)$$

where $z^{(q')} \Rightarrow (q'')t$ is a valid computation of the Mealy machine \mathcal{M} on a tape of length m .

Then, by definition of the Cartesian product,

$$(1_{\Sigma^n} \otimes [n]_{\mathcal{M}}) = \{(wq'v, wuq) : (q'v, uq) \in [n]_{\mathcal{M}}, w \in \Sigma^m\}$$

and

$$([m]_{\mathcal{M}} \otimes 1_{\Sigma^n}) = \{(q''tw', zq'w') : (q''t, zq') \in [m]_{\mathcal{M}}, w' \in \Sigma^n\}.$$

Hence, by the definition of the graded composition \odot ,

$$[m] \odot [n] = \{(q''tv, zuq) : \exists q' \text{ s.t. } (q'v, uq) \in [n]_{\mathcal{M}}, (q''t, zq') \in [m]_{\mathcal{M}}\}.$$

That is, $[m] \odot [n]$ describes all valid computations on a tape of length m , followed by all valid computations on a tape of length n , subject to the intermediate state matching (the state at the boundary of cell m and cell $m + 1$). Hence, this is the set of all valid computations of \mathcal{M} on a tape of length $m + n$. \square

The restriction of the above to the special case where the transition/rewrite function is deterministic is not problematic – the **Comp** construction can be applied in exactly the same way to the category of partial or global functions on sets, since **Comp** can be applied to any monoidal category, and nothing in the above proof required the assumption of relations, rather than functions. Therefore, (partial) deterministic Mealy machines are modelled by graded monoids in the graded category $\mathbf{Comp}_{\Sigma}(\mathbf{pSet})$, and full deterministic Mealy machines are modelled by graded monoids in the graded category $\mathbf{Comp}_{\Sigma}(\mathbf{fSet})$, where \mathbf{fSet} denotes the category of globally defined functions on sets.

This then completes Steps 5 and 6 of the programme laid out in Section 3.

9.1. Finite state automata as a special case of Mealy machines

From the programme laid out in Section 3, we expect to be able to derive algebraic models of finite state automata as special cases of algebraic models of Mealy machines.

In terms of state machines, rather than algebraic models, we consider a finite state automation to be a Mealy machine \mathcal{M} , where the transition/output function

$$\circ \in \mathbf{rSet}(\Sigma \times Q, Q \times \Sigma)$$

satisfies

$$(q'y, xq) \in \circ \Rightarrow x = y.$$

This is a Mealy machine where the transition/output function always writes the symbol that has just been read. In terms of algebraic models of machines, this condition implies that

$$(q'v, uq) \in [n] \Rightarrow u = v.$$

Theorem 7. Consider a Mealy machine

$$\mathcal{M} = (Q, \Sigma, \circ \in \mathbf{rSet}(\Sigma \times Q, Q \times \Sigma))$$

that satisfies the condition given above (that is, it always writes the symbol that has just been read), and let

$$\mathcal{A} = (Q, \Sigma, * \in \mathbf{rSet}(\Sigma \times Q, Q))$$

be the same machine, described as a finite state automaton, so

$$q' \in x * q \Leftrightarrow (q', x) \in x \circ q.$$

Then the transition homomorphism $t : \Sigma^* \rightarrow \mathbf{rSet}(Q, Q)$ of \mathcal{A} may be defined in terms of the computation homomorphism $[] : \mathbb{N} \rightarrow \mathbf{Comp}_\Sigma(Q, Q)$ of \mathcal{M} , as follows:

$$t(w) = \pi_w^{-1}[n]\pi_w \in \mathbf{rSet}(Q, Q) \quad n = |w|$$

where

$$\pi_w = \{(wq, q) : q \in Q\}, \quad \pi_w^{-1} = \{(q, qw) : q \in Q\}.$$

Conversely, the computation homomorphism $[] : \mathbb{N} \rightarrow \mathbf{Comp}_\Sigma(Q, Q)$ of \mathcal{M} may be defined in terms of the transition homomorphism $t : \Sigma^* \rightarrow \mathbf{rSet}(Q, Q)$ of \mathcal{A} as follows:

$$[n]_{\mathcal{A}} = \bigcup_{w \in \Sigma^n} \{(t(w).q, w.q) : q \in Q\} \in \mathbf{Comp}_\Sigma(\mathbf{rSet})_n(Q, Q).$$

(We use the notation ‘ $v.u$ ’ to denote concatenation of v and u in Σ^* , to avoid confusion with function application).

Proof. The proof is immediate from the interpretation in terms of machine actions in both cases. □

If, in the deterministic case, we express the transition homomorphism in terms of ‘lifting the next-state function from Σ to Σ^* ’ and describe the computation relations in the same way, we may write:

- (for finite state automata) $w * q = t(w)(q)$,
- (for Mealy machines) $(w \circ q) = [n](w, q)$.

The above property then becomes (the significantly simpler)

$$q' \in w * q \Leftrightarrow (q', w) \in w \circ q.$$

However, we require the more complex notation to express how both two-way automata and Mealy machines may be considered as special cases of bounded Turing machines, as given in Subsections 10.3 and 10.4, respectively.

10. Bounded Turing machines – two-way automata with output

We demonstrate how the **Comp** construction of Definition 7 applied to an appropriate compact closed category gives a categorical interpretation of the generalisation from two-way automata to bounded Turing machines[†].

Recall our description of a non-deterministic *bounded Turing machine* as specified by the following data: an input alphabet/output alphabet Σ , two state sets Q_l and Q_r , the *left and right moving states*, and a transition/rewrite relation

$$\circ \in \mathbf{rSet}(\Sigma \times Q_l \sqcup Q_r \times \Sigma, Q_l \times \Sigma \sqcup \Sigma \times Q_r).$$

[†] We could perhaps derive models of bounded Turing machines by applying the **Int** construction to the models of Mealy machines, but this would involve a number of additional constructions, including a meaningful analogue of the trace of Subsection 4.1 for a graded monoidal category. We do not explicitly attempt this route to the generalisation; however, it is implicit in the description of Mealy machines as special cases of bounded Turing machines, as given in Section 10.4.

In the same way as for the computation relations for Mealy machines, we introduce relations that give, for a bounded Turing machine \mathcal{T} , all possible computations of \mathcal{T} on a tape of fixed length.

Definition 11. Let \mathcal{T} denote a bounded Turing machine. We define 4 relations, called the *computation relations* as follows:

- $(l'v, ul) \in [\leftarrow n-] \subseteq (Q_l \times \Sigma^n) \times (\Sigma^n \times Q_l)$
iff there exists a computation of \mathcal{T} from $u^{(l)}$ to $l'v$.
- $(lv, ru) \in [\Rightarrow n] \subseteq (Q_l \times \Sigma^n) \times (Q_r \times \Sigma^n)$
iff there exists a computation of \mathcal{T} from ${}^{(r)}u$ to $l'v$.
- $(vr, ul) \in [n \Leftarrow] \subseteq (\Sigma^n \times Q_r) \times (\Sigma^n \times Q_l)$
iff there exists a computation of \mathcal{T} from $u^{(l)}$ to $v^{(r)}$.
- $(r'v, ru) \in [-n \rightarrow] \subseteq (\Sigma^n \times Q_r) \times (Q_r \times \Sigma^n)$
iff there exists a computation of \mathcal{T} from ${}^{(r)}u$ to $v^{(r')}$.

Taken together, these relations form a single arrow in $\mathbf{Int}(\mathbf{rSet})((\Sigma^n \times Q_l, \Sigma^n \times Q_r), (Q_l \times \Sigma^n, Q_r \times \Sigma^n))$ by defining

$$[n]_{\mathcal{T}} = \begin{pmatrix} [\leftarrow n-] & [\Rightarrow n] \\ [n \Leftarrow] & [-n \rightarrow] \end{pmatrix}$$

Hence each integer n specifies an element of the graded endomorphism monoid of the object (Q_l, Q_r) in the graded category $\mathbf{Comp}_{\Sigma, \Sigma}(\mathbf{Int}(\mathbf{rSet}))$.

As in the case of two-way automata or Mealy machines, note that $[1]_{\mathcal{T}}$ is just the transition/rewrite relation for the bounded Turing machine \mathcal{T} .

10.1. The graded compact closed category $\mathbf{Comp}(\mathbf{Int}(\mathbf{rSet}))$, and algebraic models of bounded Turing machines

We wish to demonstrate that the **Comp** construction applied to algebraic models of two-way automata gives algebraic models of bounded Turing machines. Categorically, this means that we need to construct a graded category from a monoidal tensor on the category $\mathbf{Int}(\mathbf{rSet})$. However, first note that the **Int** construction is applied to the additive structure of the category \mathbf{rSet} (that is, the disjoint union), and the **Comp** construction is applied to the multiplicative structure (that is, the Cartesian product). As both the disjoint union and the Cartesian product are defined pointwise on pairs, we may explicitly give the composition of the graded category $\mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{rSet}))$, as follows.

Proposition 8. Let $F \in \mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{rSet}))_i((X, U)(Y, V))$ be specified by the matrix

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbf{Int}(\mathbf{rSet})((\Sigma^i, \Sigma^i) \times (X, U), (Y, V) \times (\Sigma^i, \Sigma^i)),$$

and let $G \in \mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{rSet}))_j(Y, Z)$ be specified by the matrix

$$\begin{pmatrix} e & f \\ g & h \end{pmatrix} \in \mathbf{Int}(\mathbf{rSet})((\Sigma^j, \Sigma^j) \times (X, U), (Y, V) \times (\Sigma^j, \Sigma^j)).$$

Then the composite $GF \in \mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{rSet}))_{i+j}((X, U)(Z, W))$ is given by the following composition in $\mathbf{Int}(\mathbf{rSet})$:

$$\begin{pmatrix} e \times 1_{\Sigma^i} & f \times 1_{\Sigma^j} \\ g \times 1_{\Sigma^i} & h \times 1_{\Sigma^j} \end{pmatrix} \cdot \begin{pmatrix} 1_{\Sigma^j} \times a & 1_{\Sigma^i} \times b \\ 1_{\Sigma^j} \times c & 1_{\Sigma^i} \times d \end{pmatrix}$$

Proof. The proof is immediate from the description of the inherited Cartesian product, and the definition of the composition in $\mathbf{Int}(\mathbf{rSet})$. \square

We may now establish the connection between the computation relations for a bounded Turing machine and the graded composition.

Theorem 9. Consider a bounded Turing machine \mathcal{T} , and let its computation relations for words of length m and n be given by $[m]_{\mathcal{T}}$ and $[n]_{\mathcal{T}}$, respectively. Then its computation relation for words of length $m+n$ is given by

$$[m+n]_{\mathcal{T}} = [m]_{\mathcal{T}} \odot [n]_{\mathcal{T}}$$

where \odot denotes the graded composition at the endomorphism monoid of (Q_l, Q_r) in the graded category $\mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{rSet}))$.

Proof. For clarity, we use $[]$ to denote $[]_{\mathcal{T}}$ in the following proof.

We first consider the case of $[\leftarrow m+n]$. By the computation for one-way read/write machines (in Theorem 6),

$$(1_{\Sigma^n} \times [\leftarrow m-])([\leftarrow n-] \times 1_{\Sigma^m}) \subseteq [\leftarrow m+n-].$$

However, we can say more; consider a bounded Turing machine computation on a word of length $m+n$, and mark the intersection between cell m and cell $m+1$ (counting from the left) by the symbol \textcircled{a} . So, our cells are numbered

$$1 \ 2 \ 3 \ \dots \ m \ \textcircled{a} \ m+1 \ \dots \ m+n$$

Now consider a computation of \mathcal{T} on this tape that starts on the right and finishes on the left, and count the number of times the read/write head passes through the point \textcircled{a} (the *crossing number*). Also assume that this computation starts in state p and ends in state q , and takes an input word yu to an output word zv . If the read/write head passes through the point \textcircled{a} once, then

$$(qzv, yup) \in (1_{\Sigma^n} \times [\leftarrow m-])([\leftarrow n-] \times 1_{\Sigma^m}).$$

Similarly, if the read/write head passes through the point \textcircled{a} three times (clearly, it cannot pass through an *even* number of times on its way from the right-hand side to the left-hand side) then we must have that

$$(qzv, yup) \in (1_{\Sigma^n} \times [\leftarrow m-])([\Rightarrow n] \times 1_{\Sigma^m})(1_{\Sigma^n} \times [m \Rightarrow])([\leftarrow n-] \times 1_{\Sigma^m}).$$

If the read/write head passes through the point \textcircled{a} five times

$$(qzv, yup) \in (1_{\Sigma^n} \times [\leftarrow m-])(([\Rightarrow n] \times 1_{\Sigma^m})(1_{\Sigma^n} \times [m \Rightarrow]))^2([\leftarrow n-] \times 1_{\Sigma^m}).$$

In general, whatever the crossing number of the point \textcircled{a} ,

$$(qzv, yup) \in (1_{\Sigma^n} \times [\leftarrow m-])(([\Rightarrow n] \times 1_{\Sigma^m})(1_{\Sigma^n} \times [m \Rightarrow]))^i([\leftarrow n-] \times 1_{\Sigma^m})$$

for some natural number i . Therefore,

$$[\leftarrow m + n -] = (1_{\Sigma^n} \times [\leftarrow m -])(([\Rightarrow n] \times 1_{\Sigma^m})(1_{\Sigma^n} \times [m \Rightarrow]))^*([\leftarrow n -] \times 1_{\Sigma^m}).$$

The symmetry of the situation gives us the dual of this with no further calculation, so

$$[-m + n \rightarrow] = ([-n \rightarrow] \times 1_{\Sigma^m})((1_{\Sigma^n} \times [\Leftarrow m])([n \Leftarrow] \times 1_{\Sigma^m}))^*(1_{\Sigma^n} \times [-m \rightarrow]).$$

Right to right movement is dealt with in a similar way; the main point to note is that if the read head does not pass through the point \textcircled{a} , then the contents of the tape to the left of the point \textcircled{a} cannot change (this is a possible interpretation of the use of the $(1_{\Sigma^m} \times _)$ operation, which ‘writes all possible words of Σ^m on the left of a word from Σ^n ’).

The case when the read head does pass through this point is then dealt with by a similar crossing number argument, to give $[m + n \Leftarrow] =$

$$([n \Leftarrow] \times 1_{\Sigma^m})$$

∪

$$([-n \rightarrow] \times 1_{\Sigma^m})(1_{\Sigma^n} \times [m \Leftarrow])(([\Rightarrow n] \times 1_{\Sigma^m})(1_{\Sigma^n} \times [m \Leftarrow]))^*([\leftarrow n -] \times 1_{\Sigma^m}).$$

Again, duality gives the left to left movement as $[\Leftarrow m + n] =$

$$(1_{\Sigma^n} \times [\Leftarrow m])$$

∪

$$(1_{\Sigma^n} \times [\leftarrow m -])([\Leftarrow n] \times 1_{\Sigma^m})((1_{\Sigma^n} \times [m \Leftarrow])([\Leftarrow n] \times 1_{\Sigma^m}))^*(1_{\Sigma^n} \times [-m \rightarrow]).$$

Comparing the above four terms with the composition of $\mathbf{Int}(\mathbf{Comp}(\mathbf{rSet}))$ gives

$$[m + n] = [m] \odot [n],$$

as required. □

This completes Step 8 of the programme laid out in Section 3.

10.2. Deterministic bounded Turing machines

If the general description of the procedures given in Section 3 is correct, there should now be no obstacle to the construction of similar computation relations for the deterministic case – we merely need to apply the same constructions using \mathbf{pSet} instead of \mathbf{rSet} as the base category.

We have seen that \mathbf{Int} can be applied to \mathbf{pSet} in Section 7.1, and we have seen that \mathbf{Comp} can be applied to \mathbf{pSet} in Section 9.1. We now merely need to demonstrate that if we have a monoid consisting of matrices of partial functions satisfying the condition given in Section 7.1, then the graded monoid given by the \mathbf{Comp} construction of Definition 5 also consists of matrices of partial functions satisfying the same condition.

However, this is almost trivial; given a bounded Turing machine where the relation

$$\circ \in \mathbf{rSet}(\Sigma \times Q_l \sqcup Q_r \times \Sigma, Q_l \times \Sigma \sqcup \Sigma \times Q_r)$$

is partial function, we can split it into a matrix of four partial functions that must satisfy:

- $[\leftarrow 1-]$ and $[1 \Rightarrow]$ have disjoint domains.
- $[-1 \rightarrow]$ and $[\Rightarrow 1]$ have disjoint domains.

That is, entries in the same column of the matrix have disjoint domains. It is then immediate that $1_X \times f$ is a partial bijection exactly when f is a partial bijection, and similarly for $f \times 1_Y$.

Hence, applying both the **Comp** and **Int** constructions will give us models in terms of partial functions, so, with little extra work, we can obtain models of deterministic bounded Turing machines as monoid homomorphisms from \mathbb{N} to graded monoids in $\mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{pSet}))$.

10.3. Representing two-way automata as special cases of bounded Turing machines

Recall the method of representing finite state automata as special cases of Mealy machines in Section 9.1. We now demonstrate that a similar method of ‘projections’ allows us to recover the global transition relations for two-way automata from the computation relations for a bounded Turing machine where the output symbol is the same as the input symbol.

Consider a bounded Turing machine \mathcal{T} , where the transition/rewrite function satisfies the conditions required for it to specify a two-way automata – that is, when the symbol written is always the symbol that has just been read. Algebraically, this is the condition that

$$(xl, l'y) \in [\leftarrow 1-]_{\mathcal{T}} \Rightarrow x = y, \quad (rx, ly) \in [\Rightarrow 1]_{\mathcal{T}} \Rightarrow x = y,$$

and similarly for $[1 \Rightarrow]_{\mathcal{T}}$ and $[-1 \rightarrow]_{\mathcal{T}}$.

Theorem 10. Let

$$\mathcal{T} = (Q, \Sigma, \circ \in \mathbf{rSet}(\Sigma \times Q_l \sqcup Q_r \times \Sigma, Q_l \times \Sigma \sqcup \Sigma \times Q_r))$$

be a bounded Turing machine satisfying the condition given above (that is, the symbol written by the machine is the same as the one that has just been read), and let

$$\mathcal{A} = (Q, \Sigma, * \in \mathbf{rSet}(\Sigma \times Q_l \sqcup Q_r \times \Sigma, Q_r \sqcup Q_l))$$

be the same machine, described as a two-way automaton.

We can then construct the global transition homomorphism

$$[\]_{\mathcal{A}} : \Sigma^* \rightarrow \mathbf{Int}(\mathbf{rSet})(Q_l, Q_r)$$

in terms of the computation homomorphism of \mathcal{T} ,

$$[\]_{\mathcal{T}} : \mathbb{N} \rightarrow \mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{rSet}))(Q_l, Q_r),$$

as follows:

For all $w \in \Sigma^n$,

$$[w]_{\mathcal{A}} = \Pi_w^{-1} [n]_{\mathcal{T}} \Pi_w$$

where Π_w and Π_w^{-1} are defined in terms of the projections π_w and π_w^{-1} of Theorem 7 by

$$\Pi_w = \begin{pmatrix} \pi_w & 0 \\ 0 & \pi_w^{-1} \end{pmatrix}, \quad \Pi_w^{-1} = \begin{pmatrix} \pi_w^{-1} & 0 \\ 0 & \pi_w \end{pmatrix}$$

(Note that this composition is taken in the category $\mathbf{Int}(\mathbf{rSet})$).

Conversely, given the global transition homomorphism $[\]_{\mathcal{J}} : \Sigma^* \rightarrow \mathbf{Int}(\mathbf{rSet})(Q_l, Q_r)$, we can construct the computation homomorphism of the bounded Turing machine \mathcal{T} , as follows:

$$[n]_{\mathcal{T}} = \bigcup_{w \in \Sigma^n} \left\{ \begin{pmatrix} (aw, wb) & (cw, dw) \\ (we, wf) & (gw, wh) \end{pmatrix} : \begin{pmatrix} (a, b) & (c, d) \\ (e, f) & (g, h) \end{pmatrix} \in [w] \right\}$$

Proof. The proof is a direct extension of Theorem 7, and is immediate from the interpretation of the algebraic models as machine computations. \square

10.4. Representing Mealy machines as special cases of bounded Turing machines

By analogy with the representation of finite-set automata as special cases of two-way automata where the right-moving part is empty (that is, using the canonical embedding of a traced monoidal category \mathbf{V} into the compact closed category $\mathbf{Int}(\mathbf{V})$), we demonstrate how Mealy machines may be represented as special cases of bounded Turing machines where the set of right-moving states is empty.

Let \mathbf{V} be a traced monoidal category with two distinct monoidal tensors, an additive \oplus , and a multiplicative \otimes , and denote the unit objects for \oplus, \otimes by $0, I$, respectively. We further assume that there exists a categorical trace Tr on \oplus , and use this data to specify an embedding of $\mathbf{Comp}_{\Sigma}(\mathbf{V})$ into $\mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{V}))$, as follows.

Definition 12. We define a full embedding ζ of $\mathbf{Comp}_{\Sigma}(\mathbf{V})$ into $\mathbf{Comp}_{\Sigma, \Sigma}(\mathbf{Int}(\mathbf{V}))$:

On objects: For all $X \in Ob(\mathbf{Comp}_{\Sigma}(\mathbf{V}))$,

$$\zeta(X) = (X, 0) \in \mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{V}))$$

in a similar way to the canonical embedding of \mathbf{V} into $\mathbf{Int}(\mathbf{V})$.

On arrows: Let $F \in \mathbf{Comp}_{\Sigma}(\mathbf{V})_n(X, Y)$ be specified by

$$f \in \mathbf{V}(\Sigma^n \otimes X, Y \otimes \Sigma^n).$$

Then

$$\zeta(f) \in \mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{V}))_n((X, 0), (Y, 0))$$

is specified by

$$\zeta(f) = (f, 1_0) \in \mathbf{Int}(\mathbf{V})((\Sigma^n, \Sigma^n) \otimes (X, 0), (Y, 0) \otimes (\Sigma^n, \Sigma^n)).$$

Let $\mathcal{M} = (Q, \Sigma, \circ)$ be a Mealy machine, as defined in Section 4, and construct the monoid homomorphism $[]_{\mathcal{M}}$ from \mathbb{N} to $\mathbf{Comp}_{\Sigma}(\mathbf{pSet})(Q, Q)$ representing the computation of \mathcal{M} , as defined in Section 9. Composing this monoid homomorphism with the embedding ζ defined above then gives a monoid homomorphism from \mathbb{N} into

$$\mathbf{Comp}_{(\Sigma, \Sigma)}(\mathbf{Int}(\mathbf{rSet}))((Q, 0), (Q, 0)).$$

Of course, the monoid $\mathbf{Int}(\mathbf{rSet})((Q, 0), (Q, 0))$ is the image of the monoid $\mathbf{rSet}(Q, Q)$ under the canonical embedding, so this gives the representation of a Mealy machine as a two-way state machine with read/write capability (a bounded Turing machine), where the set of right-moving states is empty.

This then completes the programme laid out in Section 3.

11. Summary and discussion

The aim of treating state machines in a consistent way appears to have been a fruitful one, in that it allows us to treat algebraic models in an equally consistent way. We feel justified in saying that the **Int** or **GoI** construction is the algebraic representation of allowing two-way (or forward/backward) movement, the **Comp** construction is the algebraic representation of generalising from read capability to read/write capability, and (this is already well-established) the generalisation from functions to relations is the algebraic representation of allowing non-determinism.

Since the aim of the presentation was to give models of state machines, rather than a presentation of algebraic structures *per se*, much remains to be done about the structure of the **Comp** construction, its interaction with the **Int** construction, and its interpretation in other categories. In particular, we would like to be able to say that $\mathbf{Int}(\mathbf{Comp}(\mathbf{V}))$ is isomorphic to $\mathbf{Comp}(\mathbf{Int}(\mathbf{V}))$, but giving this a precise meaning (or indeed, any meaning) appears to require substantial algebraic input. Also, no categorical interpretation has been given to the constructions that demonstrate how a read-only machine can be modelled as a special type of read/write machine, although this is surely an example of a general construction.

Another notable point is that, although the interaction of the (traced) disjoint union and the (free) Cartesian product is what generates the models of bounded Turing machines, the distributivity of \times over \sqcup does not appear to be essential to the constructions. The categorical requirements are very light, allowing for the application of the same constructions to a number of different categories – the example of partial bijective functions (and hence inverse semigroups) is hinted at throughout, and this would allow us to construct the same classes of state machines in the (partial) reversible case. This is an almost trivial extension of the algebraic structures considered (although the interpretation as state machines may be more complex). Similarly, we could hope to model (for example) machines with a limited (global) memory, or push-down (possibly two-way) automata in the same terms, by choosing the appropriate base categories.

More interestingly, there is scope for the consideration of categories where a Cartesian or multiplicative structure is traced (such as the usual trace on the tensor product of

Hilbert spaces, or the Cartesian trace on the category of relations, where, given a relation $R \in \mathbf{rSet}(X \times U, Y \times U)$, then $Tr(F) = \{(y, x) : \exists u \text{ s.t. } (yu, xu) \in F\}$, providing a ‘wave-style’ interpretation of two-way machines, in contrast to the ‘particle-style’ constructions in this paper. We refer to Abramsky and Jagadeesan (1992) for this distinction in terms of constructions in the field of linear logic and the Geometry of Interaction, and for motivation for alternative definitions of state machines.

A more speculative aim would be to attempt to construct a consistent description of transitions of quantum state machines, using categories based on (finite-dimensional) Hilbert spaces, together with the appropriate monoidal structures and notion of trace.

Finally, the logical interpretation should not be forgotten; the original motivation for the constructions used comes equally from the field of state machines, and the field of linear logic (in particular, the Geometry of Interaction).

12. Acknowledgements

There are many people who have encouraged me in developing the ideas of this paper; in particular I cannot overstate my thanks to the anonymous referees of *Mathematical Structures in Computer Science*, both for their patience in working through often erratic manuscripts and for their contribution of theoretical concepts and particular details. I am also grateful to the editor, Robert Seely, for his patience, time and encouragement during a long process.

Thanks are also due to Samson Abramsky for support and interest in the constructions developed and the notion of additive trace as the behaviour of particles in networks, to John Baez for long and helpful discussions on compact closure in significantly different contexts, to Ian France for a detailed critical reading from a non-specialist point of view, to Jonathan Hillier for the original suggestion that I extend models of two-way automata to bounded Turing machines, and to Mark Lawson for the intuition of inverse semigroups as the algebraic representation of reversible machine computations and for references to Birget’s work.

Appendix A

As noted in the definition of two-way automata, the description we gave is not exactly the same as that of Birget (1989a). Although we would (naturally) like to claim that our model is more fundamental, it is worthwhile to observe the dissimilarity, and present a method of modelling the computation of machines, as described by Birget, by machines as described in Section 3. We present the deterministic case, and leave the non-deterministic case as a simple extension.

The definition given in Birget (1989a) is as follows.

Definition 13. A deterministic two-way automaton consists of a state set $Q = Q_l \cup Q_r$, satisfying $Q_l \cap Q_r = \emptyset$, together with an input alphabet Σ , and a next-state function,

$$\circ : \Sigma \times Q \rightarrow Q.$$

The machine interpretation of this is that if we have an instantaneous configuration of the tape

$$\dots x_{i-2}x_{i-1} \stackrel{(a)}{x_i x_{i+1}} \dots,$$

the next configuration is

$$\dots x_{i-2} \stackrel{(b)}{x_{i-1}x_i} \dots \quad b \in x_{i-1} \circ a,$$

when a is a left-moving state, or

$$\dots x_{i-1}x_i \stackrel{(c)}{x_{i+1}} \dots \quad c \in x_i \circ a$$

when a is a right-moving state. The extension of this to the non-deterministic case is given in the natural way, as allowing for a ‘next set of configurations’.

The key distinction is the question of how computation proceeds on a single symbol – a tape of length 1. In our definition, the concept of a ‘single machine step’ or global clock is less apparent; we know all about the computation on a single symbol by reading off the answer from the transition function. In the definition above, if we have a left-to-right computation starting in state q and finishing in state q' (in terms of algebraic models, $(q', q) \in [-x \rightarrow]$ for some $x \in \Sigma$), we do not know whether this happened as a single left-to-right computation, or as a left-to-right-to-left-to-right computation, or whatever.

Theorem 11. Let $\mathbb{A} = (Q = Q_l \cup Q_r, \Sigma, \circ)$ be a two-way deterministic automaton, and consider $x \in \Sigma$. Define π_l and π_r to be the partial identities on the left and right moving state sets, and define the relations j and k to be the restriction of the next state function to Q_l and Q_r , respectively.

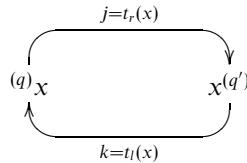
We can then construct the next-state function (as defined in Definition 3) in terms of the global transition relations, using a version of Girard’s original *resolution formula* for cut-elimination in linear logic:

$$[x] = Res(U, \sigma) = \begin{pmatrix} \pi_l & 0 \\ 0 & \pi_r \end{pmatrix} \begin{pmatrix} k & 0 \\ 0 & j \end{pmatrix} \left[\begin{pmatrix} 0 & \pi_r \\ \pi_l & 0 \end{pmatrix} \begin{pmatrix} k & 0 \\ 0 & j \end{pmatrix} \right]^* \begin{pmatrix} \pi_l & 0 \\ 0 & \pi_r \end{pmatrix}.$$

Proof. First note that a direct calculation will give

$$Res(U, \sigma) = \begin{pmatrix} 1_{Q_l} k(jk)^* 1_{Q_l} & 1_{Q_l} (kj)(kj)^* 1_{Q_r} \\ 1_{Q_r} (jk)(jk)^* 1_{Q_l} & 1_{Q_r} j(kj)^* 1_{Q_r} \end{pmatrix}.$$

Conversely, the ‘single-step’ transitions of a two-way automaton on the symbol x can be represented as follows:



All possible transitions from configurations of the form ${}^{(q)}x$ to configurations of the form $x^{(q')}$ are given by $kj, (kj)^2, (kj)^3, \dots$. Hence, $[\Rightarrow x]$ is the intersection of $(kj)(kj)^*$ with

$Q_l \times Q_r$, and so $[\Rightarrow x] = 1_{Q_l}(kj)(kj)^*1_{Q_r}$. This is the top right entry of $\text{Res}(U, \sigma)$, as required. Similar considerations will give the other three relations, $[\leftarrow x-]$, $[x \Leftarrow]$ and $[-x \rightarrow]$, as the top-left, bottom-left, and bottom-right entries, respectively. \square

Although there is no common description of Bounded Turing machines where the machine head is placed between cells of the input/output tape, similar considerations apply, in that the entire computation of the machine on a tape of length 1 is described as a single step.

However, if we describe a bounded Turing machine in similar terms to the above description of a two-way automaton, the set of computations on a tape of length 1 can be described in a similar way:

Let $\mathcal{T} = (Q = Q_l \cup Q_r, \Sigma, \circ : \Sigma \times Q \rightarrow \Sigma \times Q)$ be a deterministic bounded Turing machine, described in a similar way to the above description of a two-way automata (the intended machine semantics should be apparent). We define a two-way automaton $\mathcal{A}_{\mathcal{T}}$ that has a single character, say I , for its input alphabet, has $Q \times \Sigma$ as its set of states – we say that a state (q, x) is left-moving iff q is a left-moving state in \mathcal{T} , and similarly for the definition of the right-moving states.

It is trivial from the construction of this two-way automaton that there is a bijection between the step-by-step configurations of \mathcal{T} and those of $\mathcal{A}_{\mathcal{T}}$.

Hence, if we take the two-way automaton $\mathcal{A}_{\mathcal{T}}$, and use the method of Theorem 11, we can construct the computation relation $[1]_{\mathcal{T}}$ in the same way.

The process of constructing algebraic models of two-way machines can be thought of as a refinement of the ‘forgetting about the global clock’ process we have used above, in that we are losing information about the configurations between starting the computation and finishing the computation (including how many of them there are). More mathematically, the general procedure may be described as ‘using the trace and the **Comp** construction to eliminate the description of the behaviour of state machines on intermediate configurations’. This is also the meaning behind the requirement of partiality in the definition of determinism – an input that leads to non-terminating behaviour of a state machine is merely an input for which the corresponding algebraic model is not defined.

References

- Abramsky, S. (1996) Retracing some paths in Process algebra. In CONCUR 96. *Springer-Verlag Lecture Notes in Computer Science* 1–17.
- Abramsky, S. and Jagadeesan, R. (1992) New Foundations for the Geometry of Interaction. *Proc. Seventh IEEE Symposium on Logic in Computer Science* 211–222.
- Birget, J.-C. (1989a) Basic Techniques for Two-way Finite Automata. In: Pin, J.E. (ed.) *Formal Properties of Finite Automata and Applications. Springer-Verlag Lecture Notes in Computer Science* 56–64.
- Birget, J.-C. (1989b) Concatenation of inputs in a two-way automaton. *Theoretical Computer Science* **63** 141–156.
- Cohen, D. (1991) *Introduction to Computer Theory*, J. Wiley and Sons Inc.
- Girard, J.-Y. (1987) Linear Logic. *Theoretical Computer Science* **50** 1–102.

- Girard, J.-Y. (1988a) Geometry of interaction 1. *Proceedings Logic Colloquium '88*, North Holland 221–260.
- Girard, J.-Y. (1988b) Geometry of interaction 2. In: Martin-Löf, P. and Mints, G. (eds.) Proceedings of COLOG 88. *Springer-Verlag Lecture Notes in Computer Science* **417** 76–93.
- Howie, J. (1991) *Automata and Languages*, Oxford Science Publications, Oxford University Press.
- Joyal, A., Street, R. and Verity, D. (1996) Traced Monoidal categories. *Math. Proc. Camb. Phil. Soc.* 425–446.
- Kelly, G. and Laplaza, M. (1980) Coherence for compact closed categories. *Journal of Pure and Applied Algebra* **19** 193–213.
- Mac Lane, S. (1971) *Categories for the working mathematician*, Springer-Verlag.
- Shepherdson, J.C. (1959) The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development* 115–125.