Abstract and Concrete Models for Recursion

Martin HYLAND
DPMMS, CMS, Wilberforce Road, Cambridge CB3 0WB, U.K.

Abstract. We present a view of recursion in terms of traced monoidal categories. We sketch relevant abstract mathematics and give examples of the applicability of this point of view to various aspects of modern computer science.

Keywords. Finite automata, Flow diagrams, Fixed points, Categories, Traces

Introduction

I hope that this account of the material from my lectures at the Marktoberdorf 2007 Summer School is sufficiently self-contained to make it possible for those not present to learn from it. There is certainly little overlap with the notes I produced before the lectures; and the relation to the slides which I used for the lectures and which can be found on the Summer School website is not close either. The earlier material contains some inexact formulations, and I have attempted to make things more precise where I can. It is best to think of what is presented here as another reworking of some basic material.

The idea of recursion is central to programming. For example a subroutine may be called many times in the running of a programme: each time it is called it does what is in some sense the same thing (though hopefully to different initial data). The idea that recursion amounts to repeating the same thing over and over is familiar enough from a range of computing practice. These lectures will introduce an abstract mathematical way to think about this basic phenomenon. The examples will probably be quite familiar, but we shall look at them in a new way.

There are many abstract approaches to recursion. I have chosen to concentrate on one which is both abstract and of considerable generality, but which reflects current concrete practice. The focus will be on the idea of what is called a trace on a symmetric monoidal category. The definition is relatively recent: the original paper, treating a more general situation than we need, is [16].

Ideas drawn from abstract mathematics can seem far from concrete practice, but often the apparent distance is illusory. Typical diagrammatic methods in computing (for example wiring diagrams) reflect free categories with structure, and this makes an immediate connection between the concrete and the abstract. (A precise mathematical treatment of relevant notions of free structure is given in [11] and the treatment is further extended in [12].) For the purpose of this paper one can see the situation as follows. On the one hand the abstract notion of trace introduced to analyse recursion can be given

a concrete diagrammatic representation; on the other many diagrams with feedback are best analysed in terms of a trace

The notion of a traced monoidal category is treated in a computer science context in the book [9] by Hasegawa. That also discusses diagrams for which I have no space here. Unfortunately the book is out of print, but some information can be extracted from the author's home page. Another source of ideas but from a different perspective is [23]. The magnum opus [4] by Bloom and Esik deals in effect with traced monoidal categories with good properties, though from a point of view quite different from mine. The wealth of material is daunting but it is a valuable reference. Generally there is nothing in the literature which provides ideal background for what I say here. I have tried to do without too many prerequisites. However there is no avoiding some category theory, and I have had to restrict myself to the merest sketch. Mac Lane's book [20] remains a standard reference both for the basic theory and for relevant material on monoidal categories (chapter VII). For those with no real idea of category theory the gentler introduction in [1] is recommended. The book [2] by Barr and Wells focuses on the needs of computer scientists. A third edition and much electronic material can be found on the web.

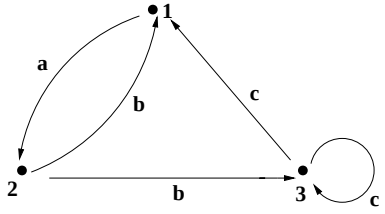
My aim has been to give a treatment of abstract material via examples. So I touch on simple mathematical examples of trace which involve relations, partial functions and permutations. I give a brief indication of how both flow diagrams from basic imperative programming and fixed points in functional programming fit into the view of recursion in terms of traces. But our leading examples will be finite automata and regular languages. This familiar material illustrates very well the flavour of the trace point of view.

There are a range of exercises, and those who hope to learn from this account should regard them as an integral part of the whole. I hope that for the most part they will be accessible to those with little background in abstract mathematics. Their organisation could probably be improved. There seems no point in trying to give an indication of difficulty. Experience at the summer school made it clear that this varied very much according to background, and also (if I may allow myself a tease) according to the standards of proof which people find acceptable.

1. Motivation: Finite Automata and Regular Languages

1.1. An example

Let us start by looking at a typical small finite automaton.



We regard 1 as the initial state and 3 as the terminal state. Then with the usual conventions, the automaton accepts the regular language $a(ba)^*b(c(ab)^*)^*$. This fact is not entirely obvious. It arises from a matrix identity. To be in accord with the usual notation for functions and matrices I write this as

$$\begin{pmatrix} 0 & b & c \\ a & 0 & 0 \\ 0 & b & c \end{pmatrix}^* = \begin{pmatrix} (c^*ba)^* & (c^*ba)^*c^*b & (c^*ba)^*cc^* \\ (ac^*b)^*a & (ac^*b)^* & (ac^*b)^*acc^* \\ ((ba)^*c)^*b(ab)^*a & ((ba)^*c)^*b(ab)^* & ((ba)^*c)^* \end{pmatrix} .$$

The bottom left hand entry, the entry in the $(3, 1)$ place, is the reverse of the regular of the regular language accepted. (That we get the reverse just comes from the change of convention.)

What we have here is an example of something familiar to most computer science students: the relation between finite automata and regular languages. Just because of it is so well known[1] it seems best to use this relation as the main focus of these notes. So let us start by reviewing some of this well known material.

1.2. Finite automata

The traditional approach is as follows. One is given a finite alphabet Σ. Then a finite automaton on the alphabet Σ is given by a finite set of states Q and an action $\Sigma \times Q \to Q$. The key issue is what sort of action to take. We shall see that a particularly judicious choice is to take non-deterministic automata and allow internal silent actions or moves: that is, arrows are marked either by a letter of the alphabet Σ or by 1 (or τ in Milner's notation) signifying the silent action.

Definition 1 *A finite non-deterministic automaton on the alphabet Σ is given a finite set of states Q and together with the action which is a relation $(1 + \Sigma) \times Q \to Q$.*

When the action is just a relation $\Sigma \times Q \to Q$, we have the usual notion of non-deterministic automaton. If further such an action is a partial function then the automaton is *deterministic*. The case of automata with silent actions but which are otherwise deterministic is also important. (There is nothing wrong mathematically with the restricted case where the action is total, but total functions generally are difficult from the point of view of recursion.)

Let Σ^* be the set of words in Σ. Mathematically it is the free monoid on Σ (the monoid operation is concatenation) and so one readily extends the action to a monoid action, which is again a relation $\Sigma^* \times Q \to Q$. Given an initial state q_0 and some terminal states t_i we are in the usual situation: we say that a word w is accepted or recognized by the machine if $w.q_0$ is related to some terminal t_i. In this context a set of words is called a language and we consider the languages recognized by finite automata, the *recognizable languages*.

[1] Before the summer school participants are asked to fill in a questionnaire indicating their degree of knowledge of various topics involved in the lectures. This was the one about which most students felt they had good knowledge, and the only one about which all knew something.

1.3. Regular languages

Let Σ be a finite alphabet and Σ^* the collection of finite words from Σ. Alternatively $\Sigma^* = \mathrm{List}(\Sigma)$ is the set of finite lists. We call a subset of Σ^* a language, and now denote such languages by a, b and so on. The collection $P(\Sigma^*)$ of languages has algebraic structure given by the following collection of operations.

- A constant zero 0 which is the empty set of words.
- A binary operation of sum $a + b$ given by the union $a \cup b$.
- A constant, the unit 1, which is the set containing just the empty word.
- A binary operation of multiplication ab given by $\{xy | x \in a \text{ and } y \in b\}$, that is, by elementwise concatenation of words from a and from b
- A unary operation, the star a^* which is the infinite union $1 + a + a^2 + \cdots$, that is, the collection of all finite concatenations of words from a.

We postpone discussion of the properties of these operations to sections 8 and 10.1, but we use them here to define the notion of regular language.

Definition 2 *The family of regular languages or regular events is the least family of languages containing the singleton letters from Σ and closed under the above operations.*

The fact that every regular language is recognizable follows by induction on the basis of the following familiar exercises. (Depending on your background you may or may not be used to using silent actions to help here. If you are not used to this point of view note that the reduction of non-deterministic to deterministic automata using the power set goes through. That is relevant to the exercises below.)

1.4. Kleene's Theorem

In a first year computer science course one is likely to see the following famous result of Stephen Kleene.

Theorem 1 *The languages recognizable by finite automata coincide with the regular languages or regular events.*

One direction of this equivalence, namely that regular languages are recognizable, is in a sense hands on coding[2] and so is generally felt to be the easier. I invite readers to remember their preferred proof in the exercises below. The other direction, that recognizable languages are regular, is usually taken to be the harder direction. In this paper we try to show that the two directions hang together conceptually. To make this clear we need to know some abstract mathematics. In particular we need to know that there are particular traced monoidal categories $\mathbf{Aut}$ of finite automata and $\mathbf{Reg}$ of matrices of regular langauges. Then the fact that recognizable languages are regular arises from the following.

Theorem 2 *The definition of languages by automata is implemented by a traced monoidal functor $\mathbf{Aut} \to \mathbf{Reg}$.*

[2]With our choice of notion of automaton it is very easy. This is not an accident. Think about it!

This would not perhaps be very compelling were it an isolated phenomenon. But it is not. As we say in the Introduction, diagrammatic methods pervade computer science. As well as automata of many kinds, there are circuits, flow diagrams, interactive systems, action structures and even diagrammatic methods in proof theory. One unifying point of view is that to the degree that one can glue diagrams together, one has a categorical composition. Moreover many diagrams permit feedback, giving a form of recursion. One general form of feedback is encapsulated in the idea of a traced monoidal category, and often with suitable modifications (which we illustrate in the case of finite automata) diagrams with feedback can be interpreted as maps in a traced monoidal category. So one can consider traced monoidal categories as a general setting in which to understand feedback. Before giving the mathematical background to this point of view, we briefly consider another example in the next section.

Exercise 1

1. *Show that for any finite set there is a finite automaton which recognizes it.*
2. *Formulate and prove a result to the effect that if there are no loops in an automaton, then the language recognized is finite.*
3. *Show that if a language is recognized by a non-deterministic finite automaton, then there is a deterministic finite automaton which also recognizes it.*
4. *Show that the empty set is a recognizable language and that the recognizable languages are closed under unions.*
5. *Show that the singleton containing the empty word is recognizable, and that the recognizable languages are closed under concatenation: if a and b are recognizable then so is $ab = \{xy | x \in a \text{ and } y \in b\}$.*
6. *Show that if a is recognizable then so is $a^* = \bigcup_{n \geq 0} a^n$ the set of all finite concatenations of words from a.*

2. Motivation: imperative programs and state

2.1. Imperative programming

Kleene's Theorem is concerned with understanding (indeed computing) the result of a certain restricted form of feedback. In this section we consider how a general form of feedback is used to define a general computation processes.

In 1971 when I started as a graduate student in Oxford, my supervisor Robin Gandy taught an undergraduate course in recursion theory using register machines. I think that he had the idea originally from John Shepherdson, but some version had occured to many people independently around 1960. In Gandy's treatment, register machines were officially given by numbered sequences of commands of the forms:

$$x := x + 1 \text{ goto } i\,; \quad \text{if } x = 0 \text{ then goto } i \text{; else } x := x - 1 \text{ goto } j\,.$$

(Here i and j refer to the next command to be executed.) So computing was explained in terms of a very primitive (one might say basic) kind of imperative programming. But almost immediately Gandy stopped using the official sequence of commands and started using flow diagrams. Here I define what are essentially the flow diagrams for a slight modification of the primitive language for register machines.

We suppose that we are given a finite set of locations, references or registers x, y , z ... . Our flow diagrams will be formed by linking instances of atomic nodes equipped with *input* and *output* ports. Take as the atomic nodes the following:

- for each register x, nodes $x := x + 1$ with one input and one output port;
- for each register x, nodes $x := x - 1$ with one input and one output port;
- for each register x, nodes if $x = 0$ then; else with one input port and two output ports

A flow diagram program is given by a finite collection of instances of such registers together with wirings from instances of output ports to input ports. (Usually it would come with a special start node (instruction) with just one output, and a stop node with just an input., but we would do best in effect to allow a number of start and stop nodes.)

If the flow diagram has the structure of a tree, then intuitively for any starting values in the registers, computation proceeds through the diagram without any return to a node already visited. But cycles in the diagram give feedback loops and so allow real recursion to occur.

2.2. Implementation on states

The standard interpretation of register machines is that they define partial recursive functions. This derives from a reading of machines as operating on states, and as with finite automata, this operation can also be thought of in terms of traced monoidal categories. We give the barest outline of this point of view.

A state is an assignment of natural numbers to registers: supposing there are r registers, we write $S = \mathbb{N}^r$ for the set of states. Let $\mathcal{P}$ be the collection of partial functions $\phi : S \to S$. Take a register machine M with n inputs and m outputs. The operation of the machine gives an $n \times m$ matrix $\Phi_M = (\phi_{ij})$ with entries $\phi_{ij} \in \mathcal{P}$. Here ϕ_{ij} is the partial function on states which arises when we start the machine at unput j and we emerge at output i. It is evident that the matrix has the property that the partial functions in the columns have disjoint domains. It follows that we can consider these matrices as maps in a simple *unique decomposition category*, in the sense of Haghverdi and Scott. (See [7] and [8] for ramifications of the theory of such special traced monoidal categories.) Write $\mathbf{Par}$ for the traced monoidal category just described. Now there is also a traced monoidal category $\mathbf{Flow}$ of flow diagrams, and quite analogous to the situation for finite automata we have the following.

Theorem 3 *The interpretation of register machines as operations on state is implemented by a traced monoidal functor* $\mathbf{Flow} \to \mathbf{Par}$.

A more detailed analysis of flow diagrams in what is essentially the spirit of the above discussion can be found in [21]. There is not space to develop things further. Of course the standard definition of computability by flow diagrama or register machines is well known. So the following exercises are more for contemplation than detailed work.

Exercise 2

1. *Probably you have experience of much more advanced programming, but just for fun write a program in the form of a flow diagram to calculate the Fibonacci sequence. How would you show that it does so? Can you say how many instructions are traversed in the calculation of the nth Fibonacci number?*

2. *Show that in the original programming language with numbered instructions, one can restrict the first (augment the register by one) instruction to the case*

$$i:\ x := x + 1 \text{ goto } i + 1\,.$$

What is one doing in this argument?
3. *Write out a formal explanation of the operation of a flow diagram program as an operation on states?*

3. Background: Elementary Category Theory

3.1. Categories

The definition of a category due to Eilenberg and Mac Lane arose from an attempt to make precise the sense in which the totality of mathematical structures of a given kind, together with the structure-preserving maps between them, can itself be regarded as a mathematical structure in its own right. We refer the reader to [1], to [2] and to [20] for the basic mathematical theory. The account in [2] is particularly directed towards computer science. Here we content ourselves with an informal account.

A category $\mathcal{C}$ consists of a collection $\mathrm{ob}(\mathcal{C})$ of *objects* (here denoted by uppercase letters $U, V, W, X, Y, Z, ...$) and for each pair (X, Y) of objects a collection $\mathcal{C}(X, Y)$ of *morphisms (or arrows or maps) from X to Y* (one writes suggestively $f : X \to Y$ for $f \in \mathcal{C}(X, Y)$); together with

identities $1_X = \mathrm{id}_X \in \mathcal{C}(X, X)$ for each X in $\mathrm{ob}(\mathcal{C})$,
composition $m_{X,Y,Z} = \circ : \mathcal{C}(Y, Z) \times \mathcal{C}(X, Y) \to \mathcal{C}(X, Z) \quad (g, f) \to g \circ f = gf,$

satisfying the following axioms:

- if $f \in \mathcal{C}(X, Y)$ then $f \circ 1_X = f$ and $1_Y \circ f = f$;
- $h \circ (g \circ f) = (h \circ g) \circ f$ whenever $f \in \mathcal{C}(X, Y)$, $g \in \mathcal{C}(Y, Z)$ and $h \in \mathcal{C}(Z, W)$.

It is good to have a range of examples in mind. The original motivation comes from large categories. First there are categories of sets: the category $\mathbf{Sets}$ of sets itself is the basic example, but one has also categories which give the Boolean-valued models for set theory, and at yet a further level of generality toposes [15]. Then there are categories of algebras: familiar examples are the categories of groups and of rings; more generally one has the category of T-algebras for a monad (sometimes called a triple) T. Then there are categories of spaces: the most familiar is $\mathbf{Top}$, the category of topological spaces; but there are many other notions of space, for example $[\Delta^{\mathrm{op}}, \mathbf{Sets}]$, the category of simplicial sets (see [22]), and from algebraic geometry, the category of schemes. Finally we mention some categories in computer science: Scott domains are well established and stable domains have now a substantial theory; I mention as well the more recent categories of games.

It is not only the case that collections of mathematical structures form categories, but also the case that many structures which appear in mathematics are themselves categories of some kind. This is a particularly fertile idea, which I learnt early in my career from Bill Lawvere. I do not try to survey the range of special examples, which have emerged

over the years, but give a traditional list of small categories. Preorders are categories with at most one map between any two objects. Monoids are categories with just one object. Groupoids are categories in which all maps are invertible. Finally groups are one object groupoids.

3.2. Free categories

Let $G = (E \overrightarrow{\rightarrow} V)$ be a directed graph: V is the collection of vertices, E the collection of directed edges and the two maps give source and target. The category $\mathbb{C}_G$ generated by G has as objects the set V of vertices, and as maps $A \to B$ the paths

$$A = A_1 \to A_2 \to A_3 \to \cdots \to A_n = B$$

from A to B in the graph. Identities are the trivial paths, and composition is given by concatenation of paths. This is the simplest example of a free construction in category theory. Free categories with structure play an important role in theoretical computer science, and frequently they can be constructed in just the same hands on fashion.

3.3. Functors and natural transformations

The idea of a functor arose out of the observation that in algebraic topology invariants such as the homology groups are defined in a simple uniform fashion, with the consequence that maps between spaces induce maps between the homology groups *in a natural way*. This amounts to regarding the categories as (large) structures and the constructions as structure preserving maps, and so one arrives at the general notion of a functor. A functor $F : \mathcal{C} \to \mathcal{D}$ assigns to each object $X \in \mathcal{C}$ an object $F(X) \in \mathcal{D}$ and to each map $f : X \to Y$ a map $F(f) : F(X) \to F(Y)$ such that

- $F(1_X) = 1_{F(X)}$
- $F(g \circ f) = F(g) \circ F(f)$ whenever $g \circ f$ is defined.

Clearly for any category $\mathcal{C}$, there is an identity functor $1_{\mathcal{C}} : \mathcal{C} \to \mathcal{C}$; it acts as the set-theoretic identity on both objects and maps. Furthermore if $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{E}$ are functors there is a composite $G \circ F : \mathcal{C} \to \mathcal{E}$; again this is given by set-theoretic composition on both objects and maps. If we restrict to small categories in the spirit of Lawvere this gives us the (large) *category of all small categories* $\mathbf{Cat}$. This has as objects the small categories $\mathbb{C}$; and as maps, functors $F : \mathbb{C} \to \mathbb{D}$. The identities and composition are as just described.

The idea of natural isomorphisms and more generally of natural transformations was part of category theory from the beginning. Suppose that $F, G : \mathcal{C} \to \mathcal{D}$ are functors. A *natural transformation* $\alpha : F \to G$ consists of a family of maps $\alpha_U : FU \to GU$ indexed over the objects $U \in \mathcal{C}$ such that for all maps $f : U \to V$ in $\mathcal{C}$, the diagram

$$\begin{array}{ccc} FU & \xrightarrow{\alpha_U} & GU \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle Gf} \\ FV & \xrightarrow{\alpha_V} & GV \end{array}$$

commutes. If all α_U are isomorphism, then α is a *natural isomorphism*.

If $\mathbb{C}$ and $\mathbb{D}$ are small categories, then $[\mathbb{C}, \mathbb{D}]$, with objects the functors from $\mathbb{C}$ to $\mathbb{D}$ and with maps the natural transformations, is itself a small category.

Exercise 3

1. *Show that $\mathbb{C}_G$ is the free category generated by G in the sense that any graph homomorphism from G to the underlying grapg of some category $\mathbb{D}$ extends uniquely to a functor from $\mathbb{C}_G$ to $\mathbb{D}$.*
2. *(a) What is the free category on the unique graph of the form $(0 \overrightarrow{\longrightarrow} 1)$?*
   (b) What is the free category on the unique graph of the form $(0 \overrightarrow{\longrightarrow} 1)$?
   (c) What is the free category on the graph $(1 \overrightarrow{\longrightarrow} 2)$ where the source and target are distinct?
   (d) What is the free category on the graph $(\mathbb{N} \overrightarrow{\longrightarrow} \mathbb{N})$ with source the identity and target the successor?
3. *Show that* Cat *has products. Show further that we have a natural isomorphism*

$$\mathbf{Cat}(\mathbb{C} \times \mathbb{D}, \mathbb{E}) \cong \mathbf{Cat}(\mathbb{C}[\mathbb{D}, \mathbb{E}]),$$

   so that $\mathbf{Cat}$ *is a cartesian closed category. (Hence it models the typed lambda calculus, see for example [19].)*

4. Background: Symmetric monoidal categories

4.1. Intuition and examples

We refer to Mac Lane [20] for the notion of a monoidal and of a symmetric monoidal category. Here we give an informal description. A *monoidal category* is a category $\mathcal{A}$ equipped with

- a tensor functor $\otimes : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$
- a choice of object $I \in \mathcal{A}$

making $\mathcal{A}$ a monoid in a suitable up to isomorphism sense. That means that we have natural isomorphisms

$$a_{UVW} : (U \otimes V) \otimes W \longrightarrow U \otimes (V \otimes W),$$

$$l_U : I \otimes U \longrightarrow U \text{ and } r_U : U \otimes I \longrightarrow U,$$

rather than equalities; and for good sense, these natural isomorphisms should satisfy coherence conditions. However there is a precise sense (the Mac Lane Coherence Theorem) in which one can replace such a monoidal category by one in which we do have equality. Such monoidal categories are called *strictly associative* or just *strict*. Most of the examples with which we are concerned are strict or have obvious strict replacements. A monoidal category is *symmetric* if it is equipped with a symmetry that is, a natural isomorphism

$$c_{UV} : U \otimes V \longrightarrow V \otimes U$$

with $c^2 = 1$, again satisfying coherence conditions. The symmetry isomorphism is very seldom an identity.

I first run through some simple general sources of symmetric monoidal categories. These require the knowledge of very elementary category theory.

1. Categories with finite products are symmetric monoidal. A choice of terminal object and of binary products gives the monoidal structure.

2. Dually, categories with finite coproducts are symmetric monoidal. A choice of initial object and of binary coproducts gives the monoidal structure.

3. Categories with biproducts (which we treat briefly in section 8 but see [20] for details) are symmetric monoidal. In this case the monoidal structure is both a product and coproduct.

4. If T is any commutative algebraic theory, then the category of T-algebras automatically has a tensor product.

It is good to have a couple of very specific examples.

5. The category $\mathbf{Rel}$ of (finite) sets and relations has a tensor product given by coproduct (disjoint union) of sets. This is an example of a category with biproducts.

6. The category $\mathbf{Rel}$ of (finite) sets and relations has a tensor product given by product of sets. In $\mathbf{Rel}$ this is neither a categorical product or coproduct. This is an example of a subcategory of a category of algebras for a commutative theory - the theory of complete $\bigvee$-lattices.

4.2. The free symmetric monoidal category

We describe the free symmetric monoidal category on an object. (Technically we are considering the strictly associative version of this notion.) The category $\mathbf{Perm}$ (for permutations) has as objects the natural numbers 0, 1, ... The maps from n to n are the elements of the symmetric group S_n with their usual composition; there are no maps n to m when $n \neq m$. The tensor product is given by $n + m$ on objects with the obvious extension to maps. The braid relations

$$s_i.s_{i+1}.s_i = s_{i+1}.s_i.s_{i+1} \text{ for } 1 \leq i \leq n-2; \qquad s_i.s_j = s_j.s_i \text{ for } |i-j| > 1,$$

where s_i is the transposition $(i\, i+1)$, enforce the coherence of the obvious symmetry $c_{n,m} \in S_{n+m} = \mathbf{Perm}(n+m, n+m)$.

The monoidal categories are a rich area of study. There are many more identifications of free such structures relevant to computer science. An early and important one is the (augmented) simplicial category originally from topology: the characterization by generators and relations in [22] comes from a characterization as a free monoidal category. There is also a wealth of material relating monoidal and symmetric monoidal categories to higher dimensional category theory.

4.3. The category of automata

We want to explain how to give a category $\mathbf{Aut}$ whose maps correspond to finite automata (on a fixed language Σ). As in the case of the free symmetric monoidal category **Perm**, we take the objects of **Aut** to be the natural numbers. As maps from n to m it is natural to take finite automata with n distinct (numbered) input states and m distinct (distinct) output states. (However we do not ask the inputs to be distinct from the outputs.) We can almost do what we want, but in fact we need to take the automata modulo contraction of insignificant silent actions. We quickly explain this notion. Let us say that a state is a *sink* if no actions lead from it and a *source* if no actions lead too it. A silent action is *insignificant* if it leads from a state which becomes a sink when it is deleted to a state which becomes a source when it is deleted.

The key point is that in $\mathbf{Aut}$, the composition of $A : n \to m$ and $B : m \to p$ is the automaton obtained from A and B taken disjointly by adding a silent action from each output of A to the corresponding input of B. To have identities for this composition, we need to contract some silent actions[3] and these actions are insignificant. (Then it does not matter whether we give identities $n \to n$ by drawing n silent actions or simply by identifying input and output nodes with no actions.)

The category $\mathbf{Aut}$ has a very easy tensor product. On objects we take addition as before and on maps the disjoint union of finite automata. Symmetries are represented by trivial automata in the same style as identities.

Exercise 4

1. *Suppose that $\mathcal{A}$ is a symmetric monoidal category. Note that any $\mathcal{A}(A, A)$ is a monoid under composition. Show that the monoid $\mathcal{A}(I, I)$ is commutative.*
2. *Suppose that $\mathcal{A}$ is a category with (chosen) finite products. Show how to define the structure a, l and r of a monoidal category on $\mathcal{A}$. Look up the axioms for a monoidal category and show that they hold.*
3. *The example $\mathbf{Perm}$ above is the free symmeric monoidal category generated by an object. What does this mean? What is needed to prove it?*
4. *Look up the coherence diagrams for a symmetry and show that they hold for the evident symmetry on $\mathbf{Perm}$.*
5. *The objects in the free monoidal category with a particular kind of tensor product generated by a single object can be taken to be $0, 1, 2, \cdots$ as was the case for $\mathbf{Perm}$. (The object n corresponds to the n-fold tensor product of the generating object.)*
   (a) What is the free category with coproducts on an object? So what is the free category with products on an object?
   (b) What is the free symmetric monoidal category in which the unit I is initial? So what is the free symmetric monoidal category in which the unit I is terminal?
6. *(i) What becomes of our category $\mathbf{Aut}$ if we take $\Sigma = \emptyset$, the empty set?*
   (ii) What becomes of our category $\mathbf{Aut}$ if we take $\Sigma = 1$, a one element set?
7. *With a little manipulation you should be able to use the flow diagrams from Section 2.1 to give a category $\mathbf{Flow}$ along the following lines. Again take the objects*

[3] An alternative formulation is that we do not allow dangling silent actions, that is, unque silent action form an input or to an output. That gives a slightly different category. I am not sure yet which I prefer.

to be the natural numbers. We will need some dummy flow diagrams (wirings with no nodes) to represent identities. Then the basic idea is that maps from n to m are given by a flow diagram with a map from $\{1, \cdots, n\}$ to some inputs and a map from some outputs to $\{1, \cdots, m\}$. Tensor product on objects is addition as before and on maps one takes the disjoint union of flow diagrams. Represent identities and symmetries by trivial automata.

5. Special case: permutations

This section is a warm-up for the general notion of trace. We look at an instance where the computational force seems pretty trivial. (Though in fact this example is the basis for an analysis of the proof theory of multiplicative Linear Logic [6], which forms part of the celebrated Geometry of Interaction perspective.)

We consider the category $\mathbf{Perm}$ whose nonempty sets of maps are $\mathbf{Perm}(n, n) = S_n$, the finite symmetric groups for $n = 0,\ 1,\ 2\ \cdots$.

For $\sigma \in S_{n+m}$ we define the trace $\mathsf{tr}_m(\sigma) \in S_n$ of σ as follows. First we define a subsidiary function $\sigma_m : n + m \to n$ recursively by

$$\sigma_m(i) = \begin{cases} \sigma(i) & \text{if } \sigma(i) \in n, \\ \sigma_m(\sigma(i)) & \text{otherwise.} \end{cases}$$

Then we set $\mathsf{tr}_m(\sigma)(i) = \sigma_m(i)$ for $1 \leq i \leq n$; that is $\mathsf{tr}_m(\sigma)$ is the restriction of the function σ_m to n.

Exercise 5

1. *Justify the recursive definition of σ_m. On what is it a recursion?*
2. *Prove that $\mathsf{tr}_m(\sigma)$ is as required a permutation.*
3. *Show that for $\sigma \in S_k$, $\mathsf{tr}_m(\sigma)$ can be defined for $0 \leq m \leq k$ inductively in m by the formulae*

$$\mathsf{tr}_0(\sigma) = \sigma\,; \quad \mathsf{tr}_{m+1}(\sigma) = \mathsf{tr}_m(\mathsf{tr}_1(\sigma))$$

4. *How does taking the trace of a permutation affect the decomposition into cycles? How does it affect the parity of the permutation?*
5. *Suppose that $\sigma \in S_n$ and $\tau \in S_m$. We define $\sigma + \tau \in S_{n+m}$ by*

$$(\sigma + \tau)(i) = \begin{cases} \sigma(i) & \text{if } 1 \leq i \leq n, \\ \tau(i - n) + n & \text{if } n1 \leq i \leq n + m. \end{cases}$$

   What is $\mathsf{tr}_m(\sigma + \tau)$?
6. *Let $\sigma \in S_{n+m}$ and $\tau \in S_m$. Show that $\mathsf{tr}_m((1 + \tau)\sigma) = \mathsf{tr}_m(\sigma(1 + \tau))$ where $1 \in S_n$ is the identity.*
7. *Let $\gamma \in S_{2n}$ be the product of the disjoint transpositions $(i\ n + i)$ for $1 \leq i \leq n$. What is $\mathsf{tr}_n(\gamma)$.*

6. Traced monoidal categories

6.1. The definition of trace

Here I define the basic notion in terms of which we consider recursion, that of traced monoidal category. We do not need the subtleties of the braided case explained in the basic reference [16]. So for us a *traced monoidal category* is a symmetric monoidal category equipped with a trace operation

$$\frac{f : A \otimes U \to B \otimes U}{\mathsf{tr}_U(f) : A \to B}$$

satisfying elementary properties of feedback. These are as follows.

- (Domain Naturality)
  For $f : A \otimes U \to B \otimes U$ and $g : C \to A$ we have

  $$\mathsf{tr}_U(f(g \otimes \mathrm{id}_U)) = \mathsf{tr}_U(f)g\,.$$

- (Codomain Naturality)
  For $f : A \otimes U \to B \otimes U$ and $h : B \to D$ we have

  $$\mathsf{tr}_U((h \otimes \mathrm{id}_U)f) = h\mathsf{tr}_U(f)\,.$$

- (Trace Naturality)
  For $f : A \otimes U \to B \otimes V$ and $k : V \to U$ we have

  $$\mathsf{tr}_U((\mathrm{id} \otimes k)f) = \mathsf{tr}_V(f(\mathrm{id} \otimes k))$$

- (Action)
  For $f : A \otimes I = A \to B \otimes I = B$,

  $$\mathsf{tr}_I(f) = f$$

  and for $f : A \otimes U \otimes V \to B \otimes U \otimes V$,

  $$\mathsf{tr}_V(\mathsf{tr}_U(f)) = \mathsf{tr}_{U \otimes V}(f)$$

- (Independence)
  For $f : A \otimes U \to B \otimes U$ and $g : C \to D$

  $$\mathsf{tr}_U(g \otimes f) = g \otimes \mathsf{tr}_U(f)$$

- (Symmetry)
  For $c_{U,U}$ the symmetry on U

  $$\mathsf{tr}_U(c_{U,U}) = \mathrm{id}_U\,.$$

I have followed my own private preferences in changing the names of some of these axioms. I regard the first three conditions as all instances of naturality. My Traced Naturality is often called Dinaturality. What I call Action is usually and oddly called Vanishing. My Independence is otherwise Superposing. The Symmetry Axiom is usually called Yanking which has at least a good diagrammatic sense. A useful perspective on the axioms is provided by Hasegawa [9]. He also gives diagrams (without the braidings in [16]) without which the axioms are hard to understand. I drew pictures in the lectures, and the slides can be found on the conference website, but I do not have enough space here.

6.2. The free compact closed category

It is a commonplace amongst workers in Linear Logic that traced monoidal categories provide a backdrop to Girard's Geometry of Interaction. This rests on a construction which was the main result of the original paper [16].

If $\mathbf{C}$ is a traced monoidal category, then its integral completion $\mathrm{Int}(\mathbf{C})$ is defined as follows.

- The objects of $\mathrm{Int}(\mathbf{C})$ are pairs (A_0, A_1) of objects of $\mathbf{C}$.
- Maps $(A_0, A_1) \to (B_0, B_1)$ in $\mathrm{Int}(\mathbf{C})$ are maps $A_0 \otimes B_1 \to B_0 \otimes A_1$ of $\mathbf{C}$.
- Composition of $f : (A_0, A_1) \to (B_0, B_1)$ and $g : (B_0, B_1) \to (C_0, C_1)$ is given by taking the trace $\mathsf{tr}_(\sigma); f \otimes g; \tau)$ of the composite of $f \otimes g$ with the obvious symmetries

$$A_0 \otimes C_1 \otimes B_0 \otimes B_1 \xrightarrow{\sigma} A_0 \otimes B_1 \otimes B_0 \otimes C_1 \,,$$

  and

$$B_0 \otimes A_1 \otimes C_0 \otimes B_1 \xrightarrow{\tau} C_0 \otimes A_1 \otimes B_0 \otimes B_1 \,.$$

- Identities $(A_0, A_1) \to (A_0, A_1)$ are given by the identity $A_0 \otimes A_1 \to A_0 \otimes A_1$.

To understand the basic result, you need to know that a compact closed category is a symmetric monoidal category in which all objects have duals. Then the following is proved in [16].

Theorem 4 *Suppose that* $\mathbb{C}$ *is a traced monoidal category. Then* $\mathrm{Int}(\mathbb{C})$ *is a compact closed category. Moreover* Int *extends to a* 2*-functor left biadjoint to the forgetful* 2*-functor from compact closed categories to traced monoidal categories.*

In the sense of this theorem $\mathrm{Int}(\mathbb{C})$ is the free compact closed category generated by the traced monoidal category $\mathbb{C}$.

Exercise 6

1. *Show that what we defined in Section 5 is a trace on the category* $\mathbf{Perm}$*. (We already checked some axioms.)*
2. *Show that the category* $\mathbf{Aut}$ *whose maps are finite automata has a trace.*
3. *Show that the category* $\mathbf{Flow}$ *whose maps are flow diagram programs has a trace. (This assumes that you have completed an earlier exercise.)*
4. *Does the category* $\mathbf{Sets}$ *have a trace? (If you have trouble consider Section 7.)*

5. *(i) Show that the monoidal category of finite sets and relations with $+$ as tensor product has a trace. Is it unique? (If you have trouble consider Section 9.)*
   (ii) Show that the monoidal category of finite sets and relations with $\times$ as tensor product has a trace. Is it unique?
6. *(i) Does the free category with products generated by an object have a trace? If it does is it unique?*
   (ii) Does the free symmetric monoidal category with I terminal generated by an object have a trace? If it does is it unique?
7. *A trace is said to be uniform just when it satisfies the following condition. Whenever*

$$\begin{array}{ccc} A \otimes X & \xrightarrow{f} & B \otimes X \\ \downarrow{\scriptstyle A \otimes h} & & \downarrow{\scriptstyle B \otimes h} \\ A \otimes Y & \xrightarrow[g]{} & B \otimes Y \end{array}$$

   commutes, then $\mathsf{tr}_X(f) = \mathsf{tr}_Y(g)$. Can you find an example of a uniform trace?
8. *Show that any compact closed category is equipped with a trace, and prove the above theorem.*
9. *Show that the trace on a compact closed category is essentially unique. (Which earlier question does this answer?)*

7. Traced monoidal categories with products

The notion of a category with products is easy and accessible and we do not give details here. The interesting feature for us is that a general form of functional programming is based on the idea of the (least) fixed point of functionals. In this section we connect that idea with the idea of a trace.

7.1. Traces and fixed points

We consider the special case of a symmetric monoidal category where the tensor product is a categorical product. Write $\Delta = \Delta_A : A \to A \times A$ for the standard diagonal map.

Suppose first that in such a category we have a trace operation

$$\frac{f : A \times U \to B \times U}{\mathsf{tr}_U(f) : A \to B}$$

We derive from it an operation

$$\frac{f : A \times B \to B}{\mathsf{fix}_B f : A \to B}$$

by setting

$$\mathsf{fix}_B f = \mathsf{tr}_B(\Delta_B.f)\,.$$

Proposition 1 *The operation* fix *is a natural parametrised fixed point operation. That is it satisfies the following.*

- *(Fixed Point Property)*
  For $f : A \times B \to B$*,*

$$f.(A \times \mathsf{fix}_B f).\Delta_A = \mathsf{fix}_B f$$

- *(Naturality)*
  If $f : A \times B \to B$ *and* $g : C \to A$ *then*

$$\mathsf{fix}_B(f(g \times B)) = (\mathsf{fix}_B f)g\,.$$

- *(Second Naturality)*
  If $f : A \times B \to C$ *and* $g : C \to B$ *then*

$$\mathsf{fix}_B gf = g.\mathsf{fix}_C f(A \times g)\,.$$

- *(Diagonal)*
  For $f : A \times B \times B \to B$

$$\mathsf{fix}_B(\mathsf{fix}_B f) = \mathsf{fix}_{B\times B}(\Delta_B.f)\,.$$

When dealing with fixed points the variable-free categorical notation becomes intolerable. When $f : A \times B \to B$ we show the variables by writing $f(a, b)$. And then $\mathsf{fix}_B f : A \to B$ can be written $\mu b.f(a, b)$. This notation presupposes the simple form of Naturality. With it the other equations (Fixed Point Property, Second Naturality, Diagonal) in the last proposition become the following.

$$f(a, \mu b.f(a, b)) = \mu b.f(a, b))$$

$$\mu b.g(f(a, b)) = g(\mu c.f(a, g(c)))$$

$$\mu b_1.\mu b_2 f(a, b_1, b_2) = \mu b.f(a, b, b)$$

The proposition above has a converse. Suppose first that in a category with finite products we have a natural parametrised fixed point operation

$$\frac{f : A \times B \to B}{\mathsf{fix}_B f : A \to B}\,.$$

We derive from it an operation

$$\frac{f : A \times U \to B \times U}{\mathsf{tr}_U(f) : A \to B}$$

as follows. We write $f = (f_1, f_2)$ where $f_1 : A \times U \to B$ and $f_2 : A \times U \to U$: take $\mathsf{fix}_U(f_2) : A \to U$ and set

$$\mathsf{tr}_U(f) = (A \times \mathsf{fix}_U(f_2)).\Delta_A\,.$$

Proposition 2 *The operation* $\mathsf{tr}()$ *just defined is a trace on a category with products.*

The passage from trace to fixed point and back are inverse to one another. This general fact was established independently by the author and Masahito Hasegawa (see [9]) but equivalent observations in a different conceptual framework were made earlier by the authors of [4] and [23].

Theorem 5 *There is a bijection between traces and natural parametrised fixed point operators on a category with products.*

7.2. Functional programming

In the simplest view of functional programming we define partial functions $\phi : \mathbb{N}^k \to \mathbb{N}$. Write P_k for the poset of such functions under extension. It is a Scott in fact algebraic domain with the compact elements being the finite functions. Now in the category with objects products of the P_k and with Scott continuous maps we can take least fixed points. We check that this is a natural parametrized fixed point operator.

Suppose that $f : A \times B \to B$. For $a \in A$ define $f_a : B \to B$ by $f_a(b) = f(a, b)$. Then

$$\mu b.f(a, b) = \bigvee_n f_a^n(\bot)$$

The naturality in A is evident, and we check the other axioms. First $\mu b.f(a, b)$ is a fixed point as by continuity we have

$$f(\mu b.f(a, b)) = f_a(\bigvee_n f_a^n(\bot)) = \bigvee_n f_a^{n+1}(\bot) = \mu b.f(a, b)\,.$$

The Second Naturality equation follows by similar considerations. We have

$$g(\mu c.f(a, g(c)) = g(\bigvee_n (f_a g)^n(\bot)) = \bigvee_n g(f_a g)^n(\bot) = \bigvee_n (g f_a)^n(g\bot)$$

But then

$$\mu b.g(f(a, b)) = \bigvee_n (g f_a)^n(\bot) \leq \bigvee_n (g f_a)^n(g\bot) \leq \bigvee_n (g f_a)^{n+1}(\bot) = \mu b.g(f(a, b))$$

shows that $\mu b.g(f(a, b)) = g(\mu c.f(a, g(c))$.

Finally we wish to show the Diagonal Property. First let $\hat{b} = \mu b.f(a, b, b)$; then $\hat{b} = \mu b.f(a, \hat{b}, \hat{b})$, and is least with this property. Suppose that $b_1 \leq \hat{b}$. Then $b_2 \leq \hat{b}$, implies $f(a, b_1, b_2) \leq \hat{b}$. Arguing inductively we have $f_{a,b_1}^n(\bot) \leq \hat{b}$ for all n and so

taking sups, $\mu b_2.f(a,b_1,b_2) \leq \hat{b}$. This shows that $b_1 \leq \hat{b}$ implies $\mu b_2.f(a,b_1,b_2) \leq \hat{b}$. Repeating the inductive argument we deduce that

$$\mu b_1 \mu b_2.f(a,b_1,b_2) \leq \mu b.f(a,b,b)\,.$$

Now let $\hat{b} = \mu b_1 \mu b_2.f(a,b_1,b_2)$, so that $\mu b_2.f(a,\hat{b},b_2) = \hat{b}$ and so $f(a,\hat{b},\hat{b}) = \hat{b}$. Suppose that $b \leq \hat{b}$. Then $f(a,b,b) \leq f(a,\hat{b},\hat{b}) = \hat{b}$. Again arguing inductively we deduce that the iterates approximating $\mu b.f(a,b,b)$ are all less than or equal to $\hat{b}$. This gives an inequality the other way round, and we deduce that

$$\mu b_1 \mu b_2.f(a,b_1,b_2) = \mu b.f(a,b,b)\,.$$

Exercise 7

1. *Establish the Bekic condition for a natural parametrised fixed point operator. If* $f : A \times B \times C \to B \times C$ *then we can compute the double fixed point*

   $$\mathsf{fix}_{B\times C}(f) : A \to B \times C$$

   as follows. We write $f = (f_1, f_2)$ *where* $f_1 : A \times B \times C \to B$ *and* $f_2 : A \times B \times C \to C$. *Then* $\mu(b,c).f(a,b,c)$ *is the pair*

   $$\big(\mu b.f_1(a,b,\mu c.f_2(a,b,c)), \mu c.f_2(a,\mu b.f_1(a,b,\mu c.f_2(a,b,c)),c)\big)\,.$$

2. *Prove the two propositions above in whatever notation you prefer. (I think it is much easier to manipulate the diagrams: you could refer to the slides on the Marktoberdorf Summer School website.)*
3. *Prove the theorem above. (Again you may find it easier with diagrams.)*
4. *Here is an alternative approach to defining a trace in a category with finite products and a natural parametrised fixed point operation. Given* $f : A \times U \to B \times U$, *we make use of two instances of the first projection* $\mathsf{fst}_{A,B} : A \times B \to A$ *and* $\mathsf{fst}_{B,U} : B \times U \to B$; *and we set*

   $$\mathsf{tr}_U(f) = \mathsf{fst}_{B,U}.\mathsf{fix}_{B\times U}(f(\mathsf{fst}_{A,B} \times U)\,.$$

   Is this a (the same) trace as defined earlier?
5. *Let* $\mathcal{C}$ *be the category of complete lattices and order-preserving maps. It is a category with evident products. Show that any* $h : B \to B$ *in* $\mathcal{C}$ *has a least fixed point* $\mu b.h(b)$. *Show that the operation taking* $f : A \times B \to B$ *to* $\mu b.f : A \to B$ *is a natural parametrised fixed point operator.*
6. *Show that a symmetric monoidal category may admit more than one notion of trace.*

8. Categories with biproducts

We consider the special case where the tensor product in a traced monoidal category is a biproduct. This situation is discussed in detail in [20], but for completeness we give a sketch here.

The first important fact, which we invite the reader to establish in the exercises below, is that a category $\mathcal{C}$ with biproducts is *enriched* in commutative monoids. For the general theory of enriched categories one should consult [17]. The concrete content of the enrichment is that each hom-set $\mathcal{C}(A, B)$ is equipped with the structure of a commutative monoid (which we write additively) and composition is bilinear in that structure. It follows that for each object A its endomorphisms $\mathrm{End}_{\mathcal{C}}(A) = \mathcal{C}(A, A)$ has the structure of what is now called (following Bill Lawvere and Steve Schanuel) a rig, that is to say a (noncommutative) ring without negatives. We make the definition explicit. A *rig* is a set R equipped with

- $0 \in R$ and $(- + -) : R \times R \to R$
- $1 \in R$ and $(-.-) : R \times R \to R$

satisfying the familiar rules

- 0 and $+$ make R an (additive) commutative monoid,
- 1 and $.$ make R a (multiplicative) monoid,
- the multiplicative structure distributes over the additive.

This analysis has a kind of converse. Let R be a rig. Then there is a category $\mathbf{Mat}(R)$ with objects the natural numbers and with maps from n to m being $n \times m$ matrices with entries in R.

Theorem 6 $\mathbf{Mat}(R)$ *is a category with biproducts.*

Exercise 8

1. *Suppose that $\mathcal{A}$ is a category with biproducts.*
   (i) Show that for any objects $A, B \in \mathcal{A}$ the hom-set $\mathcal{A}(A, B)$ has the structure of a commutative monoid.
   (ii) Show that a map $A \oplus B \to C \oplus D$ can be represented by a matrix with entries from $\mathcal{A}(A, C)$, $\mathcal{A}(B, C)$, $\mathcal{A}(A, D)$ and $\mathcal{A}(B, D)$. Show further that composition of such maps is by matrix multplication.
2. *(i) Show that composition in $\mathcal{A}$ is a bilinear map of commutative monoids.*
   (ii) Deduce that for any object $A \in \mathcal{A}$, $\mathrm{End}_{\mathcal{A}}(A) = \mathcal{A}(A, A)$ is a rig.
3. *Prove that as claimed above $\mathbf{Mat}(R)$ is a category with biproducts.*
4. *What is the free category with biproducts generated by an object. (It suffices to identify the free rig on no generators. Why?)*

9. Traced Categories with biproducts

In this section we explain what it is to equip a category with biproducts with a trace in terms of rigs. Here we concentrate on the one object case, which is the only case considered in the main reference [4].

9.1. Conway rigs

We recall the notion originally studied briefly by Conway [5]) and discussed also in [4] where it is called a Conway Algebra; but as there is other algebraic structure also associated with the fertile mind of John Horton Conway we rename the structure.

Definition 3 *A* Conway Rig *is a rig A equipped with a unary operation*

$$(-)^* : A \longrightarrow B\,;\, a \to a^*$$

satisfying the two equations

$$(ab)^* = 1 + a(ba)^*b \qquad \text{and} \qquad (a+b)^* = (a^*b)^*a^*\,.$$

Theorem 7 *In a traced monoidal category* $\mathbf{C}$ *where the tensor product is a biproduct, each* $\mathrm{End}_{\mathbf{C}}(A)$ *is a Conway Rig, the operation* $(-)^*$ *being given by*

$$a^* = \mathsf{tr}(\begin{pmatrix} 0 & 1 \\ 1 & a \end{pmatrix})\,.$$

where we interpret the matrix as a map $\begin{pmatrix} 0 & 1 \\ 1 & a \end{pmatrix} : A \oplus A \to A \oplus A$ *in the obvious way.*

For a converse we restrict ourselves here to the case of a category with biproducts generated by a single object U. (The more general case is just a bit more fiddly.) Then it suffices to require that $\mathrm{End}_{\mathbf{C}}(U)$ be a Conway Rig. The point is this, though there is much checking to do. One takes the trace of a map $A \oplus C \longrightarrow B \oplus C$ given by the matrix

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

with $a \in \mathcal{C}(A,B)$, $b \in \mathcal{C}(C,B)$, $c \in \mathcal{C}(A,C)$ $d \in \mathcal{C}(C,C)$ using the natural formula

$$\mathsf{tr}(\begin{pmatrix} a & b \\ c & d \end{pmatrix}) = a + bd^*c$$

Note that setting $C = U$ and using the operation $(-)^*$ on $\mathrm{End}_{\mathbf{C}}(U)$ in the formula enables us inductively to define the trace in all cases.

Theorem 8 *Suppose that $\mathcal{C}$ is a category with biproducts generated by an object U. Then traces on $\mathcal{C}$ correspond exactly to choices of Conway Rig structures on* $\mathrm{End}_{\mathbf{C}}(U)$.

Exercise 9

1. *Show that in a traced category with biproducts we must have*

$$\mathsf{tr}(\begin{pmatrix} 1 & a \\ 1 & a \end{pmatrix}) = a^*\,.$$

2. *Show that in a traced category with biproducts we must have*

$$\mathsf{tr}(\begin{pmatrix} 1 & a+b \\ 1 & a+b \end{pmatrix}) = (b^*a)^*b^*\,.$$

3. *Construct a proof of Theorem 7 above.*
4. *Show that if R is a Conway Algebra, then so is $M_n(A)$ the $n \times n$ matrices with entries in R.*
5. *Construct a proof of Theorem 8 above.*

10. Regular Languages and Finite Automata: reprise

10.1. The category of regular languages

The most familiar Conway Rig is that of regular languages or events. Let Σ be a finite alphabet and Σ^* the collection of finite words from Σ. Alternatively $\Sigma^* = \mathrm{List}(\Sigma)$ is the set of finite lists. The collection $P(\Sigma^*)$ of subsets of Σ^* has the structure of a Conway Rig where

- the zero 0 is the empty set of words;
- the sum $a + b$ is given by union $a \cup b$;
- the unit 1 is the set containing just the empty word;
- the multiplication ab is given by $\{xy | x \in a \text{ and } y \in b\}$, that is, by elementwise concatenation of words from a and from b;
- the star a^* is $1 + a + a^2 + \cdots$, that is, the collection of all finite concatenations of words from a.

The substructure generated by the singleton languages whose only words are the letters from Σ has as elements exactly the regular languages: it gives us the Conway Rig Reg of regular languages. By Section 9 there is a traced monoidal category $\mathbf{Reg} = \mathbf{Mat}(Reg)$ with objects $0,\ 1,\ 2, \cdots$ in the usual way, and where the maps from n to m are given by $m \times n$-matrices whose entries are regular languages.

10.2. Definition by finite automata

Recall the category $\mathbf{Aut}$ with objects also the natural numbers and with maps given by automata. Note that an automaton can be presented as a matrix with entries very simple elements of Reg. (The presentation of the category $\mathbf{Aut}$ in this fashion can be thought of along the lines of the Girard's Geometry of Interaction, but that takes us too far afield.) Consider an automaton $A \in \mathbf{Aut}(n, m)$. A is itself a $k \times k$ matrix where as we have things $k \geq n, m$. We interpret A as follows. We compute the $k \times k$ matrix A^*, and then restrict to the n input columns and m output rows. This gives us an $n \times m$ matrix with elements from Reg, that is a map in $\mathbf{Reg}(n, m)$. This provides us with evident data for a functor $M : \mathbf{Aut} \to \mathbf{Reg}$. And the root of Kleene's Theorem is that this all works.

Theorem 9 *$M : \mathbb{A} \to \mathrm{Reg}$ is a strong monoidal functor which preserves trace.*

Exercise 10

1. *Show that both $P(\Sigma^*)$ and Reg are Conway algebras.*
2. *Show that in $P(\Sigma^*)$ and so in Reg the following natural equations hold*

$$(a^*)^* = a^* = (a^n)^*(1 + a + \cdots + a^{n-1})$$

3. *When I took Part III of the Mathematical Tripos at Cambridge I had the good fortune to take a course by John Conway based on sections from his book [5]. He gave the following equation*

$$(a + b)^* = ((a + b)(b + (ab^*)^3)^*(1 + (a + b))(1 + ab^* + (ab^*)^2 + (ab^*)^3)$$

as an example of something valid in regular events but not provable from the above axioms for what we now call Conway Rigs and the two equations above. Check its validity. (Where did it come from?)

4. *Show that M in the theorem is indeed a strong monoidal functor and that it does preserve trace. Is there a connection between these two facts?*

11. Concluding section: Free Traced Category with Biproducts

By Section 9 to identify the free traced monoidal category with biproducts on an object U it suffices to identify the free Conway Rig on no generators. The category in question is then given by the matrices construction $\mathbf{Mat}$. Fortunately the free Conway was analyzed years ago by Conway himself, though his analysis is not widely known. In [5] Conway effectively identifies the elements of the free Conway rig on no generators: at least he gives the distinct elements. They are those in the set

$$\{n \mid n \geq 0\} \cup \{n(1^*)^m \mid n, m \geq 1\} \cup \{1^{**}\}.$$

The last set of exercises gives an indication of why this is and touches on related matters. Some of the algebraic manipulation is hard. I remark however that at the Summer School, John Harrison showed me that even the hardest, which had originally taken me a couple of days to discover, was readily found by the Prover9 system of William McCune. So in extremis download it and play!

Exercise 11

1. *(i) Show that* $1 + (1^*)^n = (1^*)^n$.
   (ii) Show that $(1^*)^n + 1^{**} = 1^{**} + 1^{**} = 1^{**}$.
   (iii) Show that $n.1^{**} = 1^*.1^{**} = 1^{**}.1^{**} = 1^{**}$.
   (iv) Show that $1^{***} = 2^* = 1^{**}$.
2. *Using the above equations and developing anything further you need, show that any element in the free Conway rig on no generators is equivalent to one of those given by Conway.*
3. *What is the algebraic structure on the elements of the free Conway Rig. (That is, determine the addition, multiplication and star tables.)*
4. *Show that the elements given by Conway are all distinct.*
5. *I have an interest in understanding classical proofs. here is a calculation in the free Conway rig coming from [14]. Compute the trace in the last four arguments of the matrix*

$$\begin{pmatrix} 0&0&1&1&1&1 \\ 0&0&0&0&1&0 \\ 1&0&0&0&1&0 \\ 0&1&0&0&1&1 \\ 1&0&1&0&0&0 \\ 0&0&2&1&0&0 \end{pmatrix}.$$

Fortunately the cited paper is full of typos and the answer given there is not correct. So there is no point in cheating.

6. *Prove that the following is true in any Conway Algebra.*

$$a^{****} = a^{***} .$$

References

[1] S. Awodey. *Category Theory*. Oxford Logic Guides **49**, Clarendon Press, Oxford, 2006.

[2] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice-Hall, 1990.

[3] N. Benton and M. Hyland. Traced Premonoidal Categories. *Informatique Théorique et Applications* 37, (2003), 273-299.

[4] S. L. Bloom and Z. Esik. *Iteration Theories.* Springer-Verlag, 1993.

[5] J. H. Conway. *Regular algebra and finite machines.* Chapman and Hall, 1971.

[6] J.-Y. Girard. Linear Logic. *Theoretical Computer Science* **50**, (1987), 1-102.

[7] E. Haghverdi.Unique decomposition categories, Geometry of Interaction and Combinatory Logic. *Mathematical structures in Computer Science* **10**, (2000), 205-231.

[8] E. Haghverdi and P. Scott. A categorical model for the geometry of interaction. *Theoretical Computer Science* **350**, (2006), 252-274.

[9] M. Hasegawa. *Models of sharing graphs. (A categorical semantics for* Let *and* Letrec.*)* Distinguished Dissertation in Computer Science, Springer-Verlag, 1999.

[10] J. M. E. Hyland. Proof Theory in the Abstract. *Annals of Pure and Applied Logic* **114** (2002), 43-78.

[11] M. Hyland and J. Power. Symmetric monoidal sketches. *Proceedings of PPDP 2000*, ACM Press (2000), 280-288.

[12] M. Hyland and J. Power. Symmetric monoidal sketches and categories of wiring diagrams. In *Proceedings of CMCIM 2003*, Electronic Notes in Theoretical Computer Science 100, (2004), 31-46.

[13] M. Hyland and A. Schalk. Glueing and Orthogonality for Models of Linear Logic. *Theoretical Computer Science* **294** (2003) 183-231.

[14] M. Hyland. Abstract Interpretation of Proofs: Classical Propositional Calculus. In *Computer Science Logic (CSL 2004), eds. J. Marcinkowski and A. Tarlecki*, Lecture Notes in Computer Science 3210, (2004), 6-21.

[15] P.T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium, Volumes 1 and 2.* Oxford Logic Guides **43** and **44**. Clarendon Press, Oxford, 2002.

[16] A. Joyal and R. Street and D. Verity. Traced monoidal categories. *Math. Proc Camb Phil. Soc.* **119** (1996), 425-446.

[17] G. M. Kelly. *Basic Concepts of Enriched Category Theory.* LMS Lecture Note Series **64**, Cambridge University Press (1982).

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

[19] J. Lambek and P.J.Scott. *Introduction to higher order categorical logic.* Cambridge Studies in Advanced Mathematics **7**, Cambridge University Press (1986).

[20] S. Mac Lane. *Categories for the working mathematician.* Graduate Texts in Mathematics **5**, Springer (1971).

[21] U. Martin, E. A. Mathiesen and P. Oliva. Hoare Logic in the abstract. *Proceeding of CSL 2006*, Lecture Notes in Computer science 4207, 501-515, Springer, 2006.

[22] J. P. May. *Simplicial objects in algebraic topology.* Van Nostrand, Princeton. 1967.

[23] G. Stefanescu. *Network Algebra.* Discrete Mathematics and Computer Science Series, Springer-Verlag, 2000.