Tracing the Man in the Middle in Monoidal Categories

Dusko Pavlovic

Tracing the Man in the Middle in Monoidal Categories

Dusko Pavlovic
Email: dusko.pavlovic@rhul.ac.uk

Royal Holloway, University of London, and University of Twente

Abstract. Man-in-the-Middle (MM) is not only a ubiquitous attack pattern in security, but also an important paradigm of network computation and economics. Recognizing ongoing MM-attacks is an important security task; modeling MM-interactions is an interesting task for semantics of computation. Traced monoidal categories are a natural framework for MM-modelling, as the trace structure provides a tool to hide what happens *in the middle*. An effective analysis of what has been traced out seems to require an additional property of traces, called *normality*. We describe a modest model of network computation, based on partially ordered multisets (pomsets), where basic network interactions arise from the monoidal trace structure, and a normal trace structure arises from an iterative, i.e. coalgebraic structure over terms and messages used in computation and communication. The correspondence is established using a convenient monadic description of normally traced monoidal categories.

1 Introduction

Computation as interaction. If computers are viewed as state machines (e.g. Turing machines, or automata), then computations are their executions, i.e. sequences of actions, and one can reason about such computations in terms of predicates over sequences of actions. Program correctness is established by proving that, for all possible executions, *bad things will not happen*, and that *good things will happen*. This is guaranteed, respectively, by the *safety* and the *liveness* properties [31, 9].

Often, however, this simple view of computation needs to be refined to capture not only abstract actions, but also *locality* of data and controls, and the *interactions* that cause data flows and control flows from one locality to another. This view of *computation as interaction* has been at the core of some later developments in program semantics [25, 5, 2, 43]. One of its clearest and most prominent expressions has been game semantics of computation [3, 28]. With the Internet and computer networks, computation as interaction pervaded everyday life, and the network became the computer [42]. Semantically, this means that computations cannot be reduced to linear sequences of abstract actions any more, i.e. that the latent information flows cannot be abstracted away. This is where security takes the center stage of computation: the new correctness

requirement become that *bad information flows do not happen* and that *good information flows do happen*. The former roughly corresponds to the *secrecy* family of security properties (e.g. confidentiality, privacy, anonymity), whereas the latter corresponds to the *authenticity* family (integrity, non-malleability...). But while the safety and the liveness properties where generally independent on each other, and in fact orthogonal (in the sense that each property can be uniquely decomposed into an intersection of a safety property and a liveness property [9]), the secrecy and the authenticity properties usually depend on each other in complex and subtle ways, since every secret needs to be authenticated, and most authentications are based on secrets. Remarkably, one of the fundamental attack patterns on authentication protocols, which often goes under the name *Man-in-the-Middle (MM)* [47, 19, 30], turns out to arise through deformations of the *copycat strategy*, as the fundamental interaction pattern, modelling buffers, and supplying the identities in the interaction categories [2, 1]. In the present paper we formalize this observation. The ultimate goal is to provide a framework to trace back the buffer deformations, and thus trace the MM attacks.

Tracing Man-in-the-Middle. We propose to apply categorical methods of semantics of interaction to security. The MM attack pattern, formalized in *cord calculus*, originally designed for protocol analysis, naturally leads to a categorical *trace structure*, generalizing the traces of linear operators in this case by means of a coalgebraic, iterative structure of the term algebra used in computation and communication. In the MM-attacks on authentication protocols, the intruder inserts himself[1] between the honest parties, and impersonates them to each other. MM is the strategy used by the chess amateur who plays against two grand masters in parallel, and either wins against one of them, or ties with both. MM is also used by the spammers, whose automated agents solve the automated Turing test by passing it to the human visitors of a free porn site, set up for that purpose [20]. MM is, in a sense, one of the dominant business model on the web, where the portals, search engines and social networks on one hand insert themselves between the producers and the consumers of information, and retrieve freely gathered information for free, but on the other hand use their position in the middle to insert themselves between the producers and the consumers of goods, and supply advertising for a fee. In security protocols, MM is, of course, an interesting attack pattern. The fact that an MM attack on the famous Needham-Schroeder Public Key (NSPK) protocol [40] remained unnoticed for 17 years promoted this toy protocol into what seemed to be one of the most popular subjects in formal approaches to security. Although the habit of mentioning NSPK in every paper formalizing security has subsided, we remain faithful to the tradition, and illustrate our MM-modeling formalism on the NSPK protocol. More seriously, though, the hope is that this formalism can be used to explore the scope and the power of the MM pattern in general, and

[1] I hope that no one will be offended by the established genderism of the *Man*-in-the-Middle terminology. For better or for worse, the "Man" is in concrete examples in the literature usually called Eve, or Carol.

in particular to formalize the idea of the *chosen protocol attack*, put forward a while ago [30], but never explored mathematically.

Background and related work. The claim of the present paper is that the structure of the MM attacks can be faithfully presented and usefully analyzed in *traced monoidal categories* [29]. The syntactic trace structure of the constructed categories is used to *trace out* the intruder, just like, e.g., the linear trace structure of complex vector spaces is used to trace out the *ancillae* in quantum systems. The central technical feature is that the trace structure of the particular MM frameworks arises from the *iterative* structure [11, 12, 39, 6] of the message algebras, which in effect resolves the term equations induced by the interactions, and thus propagates the data sent in messages. The coalgebraic nature of such iterative structures has been explained and analyzed in [8, 7], where also the further references can be found. The proposed framework for the MM interactions is built as an action category [37, 38] along the lines of [41] from the *cord calculus* for protocol analysis [22, 21, 15, 44], which was designed as a domain specific process calculus underlying an integrated development environment for security protocols [10]. More detailed explanations will be provided in the text, as the formalism is introduced.

Outline of the paper. Cord calculus is described in Sec. 2, and arranged into a suitable categorical structure. Categorical semantics of the MM-interactions is described and analyzed in Sec. 3. The example of the MM-attack on the NSPK protocol is worked out in Sec. 4. Finally, Sec. 5 discusses the presented approach and some ideas for future work. The categorical background (some of it apparently of independent interest) is presented in three appendices.

2 Cord semantics of processes

In this section we introduce cord spaces and build cord categories. Various versions of the cord formalism were used in [22, 14, 17, 16, 18, 21, 15]. It was a simple reaction-based process calculus, obtained by extending the strand space formalism [24] by variables and a substitution mechanism, capturing the information flows (e.g., in protocols where participants forward parts of a payload encrypted by someone else's public key). The current version simplifies away the particle reactions, and separates the term substitution mechanism from the partial ordering of actions. The latter part remains close in spirit to strand spaces, or to Lamport's preorders [32], which can be viewed as a predecessor of all such formalisms. Formally, all such formalisms subsume under Pratt's *p*artially *o*rdered *m*ulti*sets* (pomsets) [46, 26]. In the versions from [35, 13, 44, 36], a cord space is thus simply a pomset of actions with *localities*, i.e. distributed among distinct agents. To represent communication, the actions include sending and receiving messages. The messages are terms of a polynomial algebra, supporting variable assignment and substitution. The most recent version is in [45].

2.1 Cord spaces and their runs

Processes are built starting from abstract sets of

- **terms** $\mathcal{T}$, with enough variables $Var_{\mathcal{T}} \subseteq \mathcal{T}$,
- **agents (or locations)** $\mathcal{W}$, with $Var_{\mathcal{W}} \subseteq \mathcal{W}$ and
- **actions** $\mathcal{A}$, which comes with the constructors such as

$$\mathcal{W}^2 \times \mathcal{T} \xrightarrow{\langle - \rangle} \mathcal{A} \qquad Var_{\mathcal{W}}^2 \times Var_{\mathcal{T}} \xrightarrow{(-)} \mathcal{A} \qquad \ldots$$

  that generate at least the send actions $\langle A \to B : t \rangle$ and the receive actions $(X \to Y : z)$, and moreover other actions which a particular model may require, such as (νx), (τx), $(x = t)$, or $(t/p(x))$ [13, 10].

A **cord space** P is a map $\mathbb{L} \xrightarrow{isA} \mathcal{A} \times \mathcal{W}$, where the set of *actions* $\mathbb{L} = \mathbb{L}_P$ comes equipped with a preorder $\leq$, representing their *temporal* ordering. Recall that a preorder is a transitive and reflexive relation. The set $\mathcal{L}$ of cord spaces carries two monoid structures:

- $(\mathcal{L}, \otimes, \emptyset)$, where $P \otimes Q : \mathbb{L}_P + \mathbb{L}_Q \xrightarrow{[isA_P, isA_Q]} \mathcal{A} \times \mathcal{W}$ is the cord space over the preorder $\mathbb{L}_P + \mathbb{L}_Q$ where the actions of P remain incomparable with the actions of Q, and
- $(\mathcal{L}, \cdot, \emptyset)$, where $P \cdot Q : \mathbb{L}_P < \mathbb{L}_Q \xrightarrow{[isA_P, isA_Q]} \mathcal{A} \times \mathcal{W}$ is the cord space over the preorder $\mathbb{L}_P < \mathbb{L}_Q$ where every action of P precedes every action of Q.

In each case, $\emptyset$ represents the empty cord space $\emptyset \longrightarrow \mathcal{A} \times \mathcal{W}$. Clearly, these two operations respectively correspond to the parallel and the sequential composition of cord spaces. Repeated application of these operations to actions generates most, but not all cord spaces [26]. Given a cord space P, its sets of the receive and the send actions are

$$\mathsf{recvs}(P) = \{\ell \in \mathbb{L}_P \mid \exists XYz.\ \ell\ isA\ (X \to Y : z)\}$$
$$\mathsf{sends}(P) = \{\ell \in \mathbb{L}_P \mid \exists ABt.\ \ell\ isA\ \langle A \to B : t \rangle\}$$

A **run** of the cord space P is a map $\surd_P : \mathsf{recvs}(P) \longrightarrow \mathsf{sends}(P)$ such that

$$k = \surd\ell \Longrightarrow k \not\geq \ell$$

In other words, extending the temporal preorder by setting for every $\ell \in \mathsf{recvs}(P)$ that $\surd\ell \leq \ell$ must not introduce any new cycles.

Remark. The temporal ordering of $\mathbb{L}$ is not required to be asymmetric, because different actions $p \neq q$ may occur at the same time, and thus satisfy $p \leq q$ and $p \geq q$. With abstract actions, one could assume that such actions can be identified, or sequentialized. This is done in Pratt's pomsets (partially ordered multisets) [46]. However, when actions involve terms, as they do in the above model, and when an action may depend on another action for the values that

need to be substituted before it can be executed, then the temporal precedence loops may correspond to *deadlocks*. Effective runs, of course, need to be deadlock-free. Even if the notion of a cord space was restricted to disallow temporal loops, such loops would arise as deadlocked runs, and would need to be taken into account. E.g., a cord space with a single send and a single receive action has no runs if the sent terms depend on some received data. Cord Spaces with no runs, and with cyclic dependencies arise naturally, and the existence of effective runs cannot be imposed, but needs to be analyzed.

2.2 Cord processes

From action structures to interaction categories. The composition operations over cord spaces naturally lead to a categorical structure, as soon as the input and the output interfaces of cords are displayed. The categorical composition of the cord processes can be obtained from the sequential composition of cord spaces, whereas the parallel composition yields the monoidal structure. The resulting category can be viewed as an instance of Milner's *action structure* construction [37, 38]. This view uncovers a common structural denominator for a wide gamut of process representations. Moreover, it provides a uniform framework for the categorical abstraction operations [41], which we shall use to capture secure information flows.[2]

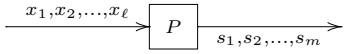
However, the cord category presented here is, strictly speaking, *not* an action structure. Although its objects, its morphisms and even its syntactic presentation are just as in an action structure, its composition is not derived from the sequential composition of processes, but rather from a minimal sequentialisation of the parallel composition. The upshot of this is that the resulting category carries a natural *trace* structure [29], in contrast with the original action structures. This structure will then be used to define the composition in a *category of interactions*. Security protocols can be specified as certain interactions, i.e. morphisms in that category; certain attacks on them then arise by composing interactions.

The idea that the composition of interactions can be characterized as a combination of *parallel composition and hiding* has previously been developed within the framework of *interaction categories* [2]. The fact that the present setting requires deviating from the sequential composition, and defining a categorical composition based on the parallel composition and hiding, in order to support the trace structure, needed for modeling interactions — can be viewed as a confirmation, and an interesting realisation of that idea.

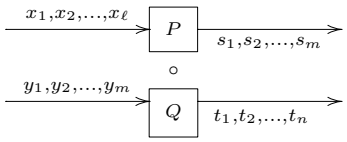
[2] Our low level syntax, with the convention that input interface and the binding operators are denoted by the round brackets $(\boldsymbol{x})$, whereas the output interface and the send action are written in the angle brackets $\langle \boldsymbol{s} \rangle$, is inherited from the action calculus. For the high level structures, though, the graphic notation for monoidal, traced and compact categories is more convenient, and will be mixed with the syntactic calculus.

Simple typing. A **cord process** consists of a cord space with input and output declarations. If we assume, for simplicity, that all data are of the same type, then an input interface declaration boils down to a tuple of distinct variables $\boldsymbol{x}_{1..\ell} = \big(x_1\ x_2\ \ldots\ x_\ell \big)$, and the output interface is a tuple arbitrary terms $\boldsymbol{s}_{1..m} = \big\langle\, s_1\ s_2\ \ldots\ s_m \,\big\rangle$. Either of these tuples can be empty, in which case we write $()$ and $\langle\rangle$. A cord process $p : \ell \longrightarrow m$ is thus viewed as an expression in the form $\big(\boldsymbol{x}_{1..\ell} \big)\, [P] \big\langle\, \boldsymbol{s}_{1..m} \,\big\rangle$, where P is a cord space. The variables $\boldsymbol{x}_{1..\ell}$ may or may not occur in (the terms of the actions of) P and in $\boldsymbol{s}_{1..m}$. More precisely, a cord process $p : \ell \longrightarrow m$ is an equivalence class of such expressions modulo variable renaming, consistently throughout P and $\boldsymbol{s}_{1..m}$.

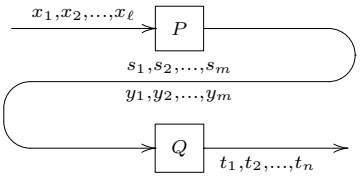
The category $\mathcal{P}$ consists of arities and the cord processes between them. It will often be convenient to extend the above action calculus syntax to diagrams, with the cord spaces enclosed in the boxes, and the interfaces displayed on the arrows:



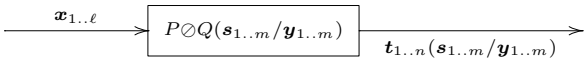
The **composition** of cord processes



can then be obtained by connecting the output interface of P with the input interface of Q



and performing the induced substitutions:



where the cord space $P \oslash Q(\boldsymbol{s}_{1..m}/\boldsymbol{y}_{1..m})$ is the minimal order extension the parallel composition $P \otimes Q$, induced by the substitution $(\boldsymbol{s}_{1..m}/\boldsymbol{y}_{1..m})$. To understand why the order needs to be extended, note that substituting, say $s_i(z)$ for y_i in an action b of Q may introduce a variable z, which may be bound to an action $a\ isA\ (z)$ in P. In such cases, we must add $a < b$ to the preorder of $P \oslash Q(\boldsymbol{s}_{1..m}/\boldsymbol{y}_{1..m})$ in order to preserve the information flow through z.

The preorder $\mathbb{L}_P \oslash \mathbb{L}_Q$ of the cord space $P \oslash Q : \mathbb{L}_P \oslash \mathbb{L}_Q \longrightarrow \mathcal{A} \times \mathcal{W}$ is thus defined over the underlying set $\mathbb{L}_P + \mathbb{L}_Q$, by setting

$$\begin{aligned} a < b \iff\ & a, b \in \mathbb{L}_P \wedge a < b \\ \vee\ & a, b \in \mathbb{L}_Q \wedge a < b \\ \vee\ & a \in \mathbb{L}_P \wedge b \in \mathbb{L}_Q \wedge \mathrm{BV}(a) \cap \mathrm{FV}(b) \neq \emptyset \end{aligned} \tag{1}$$

where $\mathrm{BV}(a)$ denotes the set of variables bound by the action α, such that $a\ isA\ \alpha$, and $\mathrm{FV}(b)$ is the set of the free variables that occur in the terms of the action β, such that $b\ isA\ \beta$.[3]

Remark. The output terms are generally in the form $s_i = s_i(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v})$, i.e. they may depend on any variables from three disjoint tuples:

- $\boldsymbol{x}$, which are bound to the input interface $\big(\, \boldsymbol{x} \,\big)$,
- $\boldsymbol{u}$, which are bound to some binding operations on the cord space P, such as (u_i), (νu_i), or $(u_i = t)$, and
- $\boldsymbol{v}$ are free.

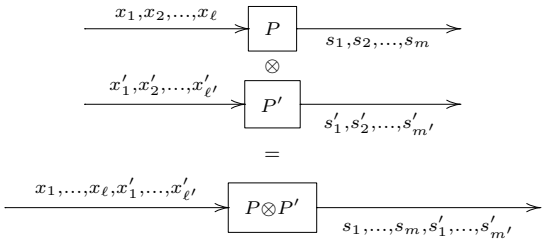
Any name clashes of any of these variables with any of the corresponding tuples of variables used in Q and t_j must be eliminated by renaming prior to the composition.

The **identities** $\mathrm{id} : n \longrightarrow n$ in $\mathcal{P}$ can be thought of as the *buffers*, i.e. the trivial cord processes that perform no processing, and just pass the inputs to the output interface. Formally, they are the expressions in the form $\big(\, x_1 \ \ldots\ x_n \,\big)\, [] \,\big\langle\, x_1 \ \ldots\ x_n \,\big\rangle$, where $[]$ is the empty cord space. In fact, every function $f : \{1, 2, \ldots, n\} \longrightarrow \{1, 2, \ldots m\}$ induces (contravariantly!) a unique cord process $\pi_f : m \longrightarrow n$, defined

$$\pi_f = \big(\, x_1\ x_2\ \ldots\ x_m \,\big)\, [] \,\big\langle\, x_{f(1)}\ x_{f(2)}\ \ldots\ x_{f(n)} \,\big\rangle$$

which just rearranges the data from the input, and displays them at the output, with no processing. If $\mathbb{N}$ is the category of finite sets $n = \{0, 1, \ldots, n-1\}$ and functions between them, then $\pi : \mathbb{N} \longrightarrow \mathcal{P}$ can be viewed as a faithful functor, identity on objects, thus displaying $\mathbb{N}$ as a subcategory of $\mathcal{P}$. As explained in [41], $\mathcal{P}$ is freely generated over $\mathbb{N}$ by adjoining variables, terms, and cord spaces.

If $\mathbb{N}$ itself is viewed as the free strictly cocartesian category (i.e. with strict coproducts) over one generator, then $\mathcal{P}$ is the free strictly monoidal category, generated by a single object, and the morphisms induced by the terms from $\mathcal{T}$, and the cord spaces from $\mathcal{L} = \mathcal{L}_{\mathcal{T},\mathcal{W},\mathcal{A}}$. The **tensor product** in $\mathcal{P}$ is induced by the parallel composition of cord spaces, and the juxtaposition of the interfaces:



[3] For a term $t \in \mathcal{T}$, the set $\mathrm{FV}(t)$ is defined by the usual inductive clauses. For an action $\alpha \in \mathcal{A}$, the set $\mathrm{BV}(\alpha)$ is just $\mathrm{BV}(x) = \mathrm{BV}(\nu x) = \{x\}$, and $\mathrm{BV}(x_1, \ldots, x_k = t_1, \ldots, t_k) = \{x_1, \ldots, x_k\}$.

where the variables from $\boldsymbol{x}_{1..\ell} = \left(x_1 \ \ldots \ x_\ell \right)$ are (if necessary, renamed to be) disjoint from the variables in $\boldsymbol{x'}_{1..\ell'} = \left(x'_1 \ \ldots \ x'_{\ell'} \right)$, and the cord space $P \otimes P'$ is the parallel composition of cord spaces, where each action of P remains incomparable with all actions of P'. Clearly, the tensor $\otimes$ in $\mathcal{P}$ extends the coproduct $+$ in $\mathbb{N}$, and boils down to it on the objects, i.e. $m \otimes n = m + n$. The tensor unit is thus the empty arity 0. The obtained monoidal structure is symmetric, and strictly associative and unitary.

Remark. An action structure over cord spaces would be a monoidal category with the same objects and morphisms as above, and even the same monoidal structure. However, the composite of $\left(\boldsymbol{x} \right) [P] \left\langle \ \boldsymbol{s} \ \right\rangle$ and $\left(\boldsymbol{y} \right) [Q] \left\langle \ \boldsymbol{t} \ \right\rangle$ would be $\left(\boldsymbol{x} \right) [P \cdot Q(\boldsymbol{s}/\boldsymbol{y})] \left\langle \ \boldsymbol{s} \ \right\rangle$, rather than $\left(\boldsymbol{x} \right) [P \oslash Q(\boldsymbol{s}/\boldsymbol{y})] \left\langle \ \boldsymbol{s} \ \right\rangle$.

Question. What is the universal property of $\mathcal{P} = \mathcal{P}_\mathcal{L}$? The general results of [41] tell that the action structure constructed over cord spaces is the free symmetric strictly monoidal category generated over a single object 1, by adjoining

- for every cord space $P \in \mathcal{L}$ an endomorphism $()[P]\langle\rangle : 0 \longrightarrow 0$, and
- for every variable x, an indeterminate arrow $()[]\langle x \rangle : 0 \longrightarrow 1$, and
- an abstraction $(x)[]\langle\rangle : 1 \longrightarrow 0$.

How does the modified composition of the cord category $\mathcal{P} = \mathcal{P}_\mathcal{L}$ change this result?

Refined typing. For simplicity, in the above description of the category $\mathcal{P}$ we ignored the issues of typing: an arity was just a tuple. In order to describe *distributed* cord processes, one must distinguish at the interfaces the entries for the terms from the entries for agent identifiers. At the very least, the input and the output interfaces thus need to be typed as products of the types $\mathcal{T}$ and $\mathcal{W}$ of terms and agents respectively. Assuming that these two types do not depend on each other in any way, the arities just split in two parts, and become pairs of natural nubmers. A cord process $p : \langle k, m \rangle \longrightarrow \langle \ell, n \rangle$ is now in the form

$$\left(\boldsymbol{X}_{1..k} \ \boldsymbol{x}_{1..m} \right) [P] \left\langle \ \boldsymbol{A}_{1..\ell} \ \boldsymbol{s}_{1..n} \ \right\rangle$$

where X_i are agent variables (roles), x_i are term variables, A_i are agent identifiers (constant or variable), and s_i are arbitrary terms.

Various aspects of distributed computation induce further refinements of the type system. E.g. local variables, available only to particular agents, are given with a map $Var_\mathcal{T} \longrightarrow \mathcal{W}$. A local variable can thus be assigned a value only after its locality is known. This means that in the input interface, it can only occur after the corresponding agent variable. The typing of terms is thus dependent on the type of agents, and we have at least one level of dependent type theory. Further type dependencies arise because

- an agent $A^\mathcal{W}[X^\mathcal{W}]$ may depend on the different roles X that she may play in a cord space,

– a term $t^{\mathcal{T}}[X^{\mathcal{W}}]$ may be computed in different ways in different roles,
– a term $t^{\mathcal{T}}[x^{\mathcal{T}}]$ may be computed in different ways depending on the outcome of the computation of some other term...

In the present paper, we shall mostly need just the dependency of terms on agents. In the rest of the paper, we denote by Γ, Φ, Ψ the general types, as the objects of $\mathcal{P}$, in contrast with the simple arities ℓ, m, n.

Contexts. Let $\boldsymbol{v} : \Gamma$ be the tuple of all free variables that occur in a cord process $p : \Phi \longrightarrow \Psi$. Then Γ is the *type context* of the cord process p. The subcategory of $\mathcal{P}$ consisting of cord processes like p, whose free variables are contained in $\boldsymbol{v}$, is denoted $\mathbb{P}[\boldsymbol{v} : \Gamma]$, or just $\mathbb{P}[\boldsymbol{v}]$. The cord processes that contain no free variables constitute the category $\mathbb{P} = \mathbb{P}[]$. It is easy to see that each $\mathbb{P}[\boldsymbol{v}]$ is closed under the monoidal structure of $\mathcal{P}$. Alternatively, $\mathbb{P}[\boldsymbol{v}]$ can be viewed as the subcategory of $\mathbb{P}$ spanned by the cord processes in the form $\left(\boldsymbol{v}\ \boldsymbol{x} \right)\ [P]\ \left\langle \boldsymbol{v}\ \boldsymbol{s} \right\rangle$.

2.3 Runs of cord processes

A run of a cord process represented by the expression $\left(\boldsymbol{x} \right) [P] \left\langle \boldsymbol{s} \right\rangle$ is an expression in the form $\left(\boldsymbol{x} \right) [P^{\surd}] \left\langle \boldsymbol{s} \right\rangle$, where $P^{\surd}$ is a run extending the cord space P, i.e. a pair

$$P^{\surd} = (P, \surd_P : \mathsf{recvs}(P) \longrightarrow \mathsf{sends}(P))$$

The category $\mathcal{R}$ of cord runs inherits all structure from the category $\mathcal{P}$ of cord processes, and comes with the obvious identity-on-the-objects forgetful functor $\mathcal{R} \longrightarrow \mathcal{P}$.

Lemma 2.1. *Let* $\surd_P : \mathsf{recvs}(P) \longrightarrow \mathsf{sends}(P)$ *and* $\surd_Q : \mathsf{recvs}(Q) \longrightarrow \mathsf{sends}(Q)$ *be runs of cord processes* $\left(\boldsymbol{x} \right) [P] \left\langle \boldsymbol{s}_{1..k} \right\rangle$ *and* $\left(\boldsymbol{y}_{1..k} \right) [Q] \left\langle \boldsymbol{t} \right\rangle$. *Then the disjoint union* $\surd_P + \surd_Q : \mathsf{recvs}(P + Q) \longrightarrow \mathsf{sends}(P + Q)$ *is a correct run of the composite process* $\left(\boldsymbol{x} \right) [P \oslash Q(\boldsymbol{s}/\boldsymbol{y})] \left\langle \boldsymbol{t}(\boldsymbol{s}/\boldsymbol{y}) \right\rangle$.

Proof. The correctness requirement of a run is that $\surd a \not\geq a$. This property will be preserved, because $\surd a$ is in the same component (P or Q) as a, and the composition $P \oslash Q(\boldsymbol{s}/\boldsymbol{y})$ only makes some Q-actions come after some P actions (that bind their variables). □

3 Cord semantics of interactions

The informal idea of *process interaction* is that two processes feed each other some data, and process them together, i.e. partially evaluate over them. Two cord processes thus interact when a part of the outputs of one of them is piped to the input interface of the other, and vice versa. This framework allows us to formalize some security questions: *What properties of the information flows*

are preserved under the interaction? Which new information flows emerge, and which old ones vanish? To formalize this idea, we partition the interfaces of each process into an Initiator and a Responder part. Two interaction programs are then composed by passing the Responder data from one to the Initiator interface of the other. These data are then propagated, and the abstract partial evaluation over them is performed using the trace structure of the cord category.

3.1 Iterations and traces of cord processes and runs

Iterative algebras and their various extensions provide a widely studied view of program execution [11, 12, 39, 6]. On the other hand, an important categorical form of iteration is given by the notion of *traced* monoidal categories [29]. In categories of programs, these two structures turn out to coincide.

Proposition 3.1. *The uniform normal trace structures on the cord category $\mathcal{P} = \mathcal{P}_{\mathcal{T},\mathcal{W},\mathcal{A}}$ are in one-to-one correspondence with the iterative structures of the term algebra $\mathcal{T}$.*

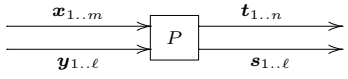
Background. The iterative structures are defined in C.2. An elegant coalgebraic account of the various versions and extensions of iterative algebras can be found in [8]. A general survey of coalgebra from this angle is provided in [7]. The general trace structures over monoidal categories are defined in [29]. The cord category naturally carries a *normal* trace structure. The normality requirement $Tr^U(f \otimes U) = f$ is shared by the traces over relations, but not by the traces over vector spaces. The importance of this requirement in the current context is that it allows a functorial presentation of the trace operations, which seems crucial for detecting the MM attacks. A convenient algebraic characterization of normal traces is given in the Appendix.

Proof. Suppose that $\mathcal{T}$ is an iterative algebra. By Def. C.2, this means it contains the iteration operation $(-)^\dagger$, which assigns to every system of $k \leq \ell$ *guarded* equations

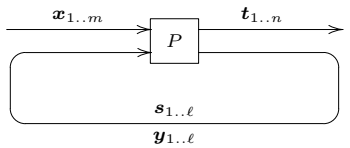
$$\begin{aligned} \boldsymbol{y}_{1..k} &= \boldsymbol{f}_{1..k}(\boldsymbol{y}_{1..\ell}) \ \text{ has a unique solution} \\ \boldsymbol{f}_{1..k}{}^\dagger(\boldsymbol{y}_{k+1..\ell}) &= \left\langle\, f_1^\dagger(y_{k+1}, \ldots, y_\ell) \ \ldots \ f_k^\dagger(y_{k+1}, \ldots, y_\ell) \,\right\rangle \ \text{ i.e.} \\ \boldsymbol{f}^\dagger{}_{1..k}(\boldsymbol{y}_{k+1..\ell}) &= \boldsymbol{f}_{1..k}\left(\boldsymbol{f}^\dagger{}_{1..\ell}(\boldsymbol{y}_{k+1..\ell})\right) \end{aligned}$$

The assumption that the equations are guarded means that the operations $\boldsymbol{f}_{1..k} = \left\langle\, f_1 \ f_2 \ \ldots \ f_k \,\right\rangle$ are not projections in the form $f_j(\boldsymbol{y}_{1..\ell}) = y_i$ for $1 \leq i \leq k$. The operations different from such projections are called *guards*.

Using the iteration $(-)^\dagger$, we define the trace functor $Tr : \mathcal{P}^{\circlearrowleft} \longrightarrow \mathcal{P}$, i.e. the trace operation $Tr^\ell_{mn} : \mathcal{P}(m \otimes \ell, n \otimes \ell) \longrightarrow \mathcal{P}(m, n)$. The idea is that the trace $Tr(p) : m \longrightarrow n$ of a program $p : m \otimes \ell \longrightarrow n \otimes \ell$, which is in the form



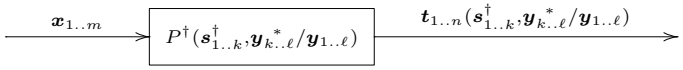
should be obtained by passing the outputs $\boldsymbol{s}_{1..\ell}$ to the inputs $\boldsymbol{y}_{1..\ell}$



However, since the variables $\boldsymbol{y}_{1..\ell}$ may occur in the terms $\boldsymbol{s}_{1..\ell}$, this substitution must be done iteratively: the terms $\boldsymbol{s}_{1..\ell}$ must also be substituted for $\boldsymbol{y}_{1..\ell}$ in themselves. This iteration can terminate if the system of equations

$$\boldsymbol{y}_{1..\ell} = \boldsymbol{s}_{1..\ell}(\boldsymbol{x}, \boldsymbol{y}_{1..\ell}, \boldsymbol{u}, \boldsymbol{v})$$

has a solution. Like before, we denote by $\boldsymbol{u}$ the variables that are bound to some binding operations in P, and by $\boldsymbol{v}$ the free variables that occur in some s_i. In general, this system may not be guarded, i.e. some of the equations may boil down to the form $y_i = y_j$. With no loss of generality, we can rearrange the system so that the first k equations $\boldsymbol{y}_{1..k} = \boldsymbol{s}_{1..k}(\boldsymbol{y}_{1..k}, \boldsymbol{y}_{k..\ell}, \boldsymbol{x}_{1..m})$ (for $0 \leq k \leq \ell$) are guarded, whereas the last $\ell - k$ equations are in the form $y_i = y_j$. Since the last $\ell - k$ equations just partition the variables $\boldsymbol{y}_{k..\ell}$, these equations can be eliminated by choosing a single representative for each equivalence class of variables. This yields a vector of variables $\boldsymbol{y}^*_{k..\ell}$, some of which may be equal. On the other hand, the assumption that $\mathcal{T}$ is an iterative algebra implies that the guarded system $\boldsymbol{y}_{1..k} = \boldsymbol{s}_{1..k}(\boldsymbol{x}, \boldsymbol{y}_{1..k}, \boldsymbol{y}^*_{k..\ell}, \boldsymbol{u}, \boldsymbol{v})$ has a unique solution $\boldsymbol{s}^\dagger_{1..k}(\boldsymbol{x}, \boldsymbol{y}^*_{k..\ell}, \boldsymbol{u}, \boldsymbol{v})$, which means that $\boldsymbol{s}^\dagger_{1..k} = \boldsymbol{s}_{1..k}(\boldsymbol{x}, \boldsymbol{s}^\dagger_{1..k}, \boldsymbol{y}^*_{k..\ell}, \boldsymbol{u}, \boldsymbol{v})$ holds. We can now define the trace program $Tr^\ell_{mn}(p) : m \longrightarrow n$ by substituting the solutions $\boldsymbol{s}^\dagger_{1..k}$ and $\boldsymbol{y}^*_{k..\ell}$ for $\boldsymbol{y}_{1..\ell}$



After these substitutions, it may occur that a variable u_i, bound in an action α of P (e.g., a receive, or nonce generation action), may be introduced in the substitution instance of $\beta(\boldsymbol{s}^\dagger/\boldsymbol{y})$ of some other action β of P. Since the variable bindings implement the information flow, α must precede β in $P^\dagger$.

The cord space $P^\dagger : \mathbb{L}^\dagger \longrightarrow \mathcal{A} \times \mathcal{W}$ is thus obtained from $P : \mathbb{L} \longrightarrow \mathcal{A} \times \mathcal{W}$ by strengthening the ordering of $\mathbb{L}$ to capture this. We define $\mathbb{L}^\dagger$ to be the preorder with the same elements as $\mathbb{L}$, and such that for all a, b holds

$$a < b \text{ in } \mathbb{L}^\dagger \iff a < b \text{ in } \mathbb{L} \ \vee \ \mathrm{BV}(a) \cap \mathrm{FV}(b) \neq \emptyset$$

where $\mathrm{BV}(a)$ and $\mathrm{FV}(b)$ are the sets of the bound and the free variables respectively, as described in section 2.2. The claim is that this defines a functor $Tr : \mathcal{P}^\bigcirc \longrightarrow \mathcal{P}$, as described in the Appendix. The grading on $\mathcal{P}$ is trivial, i.e.

$|U| = 1$ for all U. The normality requirement

$$Tr^U\left(A \otimes U \xrightarrow{f} B \otimes V \xrightarrow{B \otimes u} B \otimes U\right) = Tr^V\left(A \otimes V \xrightarrow{A \otimes u} A \otimes U \xrightarrow{f} B \otimes V\right)$$

is satisfied because for

$$f = \left(\, \boldsymbol{x}_A \;\; \boldsymbol{y}_U \,\right)[P]\left\langle\, \boldsymbol{s}_B \;\; \boldsymbol{t}_V \,\right\rangle \text{ and}$$
$$u = \left(\, \boldsymbol{z}_V \,\right)[Q]\left\langle\, \boldsymbol{t}_U \,\right\rangle$$

it is the property of iterative algebras that the same solutions of the system

$$\boldsymbol{y}_U = \boldsymbol{r}_U(\boldsymbol{z}_V)$$
$$\boldsymbol{z}_V = \boldsymbol{t}_V(\boldsymbol{x}_A, \boldsymbol{y}_U)$$

are obtained both from

$$\boldsymbol{y}_U = \boldsymbol{r}_U\left(\boldsymbol{t}_V(\boldsymbol{x}_A, \boldsymbol{y}_U)\right) \text{ and from}$$
$$\boldsymbol{z}_V = \boldsymbol{t}_V\left(\boldsymbol{x}_A, \boldsymbol{r}_U(\boldsymbol{z}_V)\right)$$

The requirement that

$$Tr^U\left(A \otimes U \xrightarrow{f \otimes U} B \otimes U\right) = \left(A \xrightarrow{f} B\right)$$

follows directly from the definition, because for $f = \left(\, \boldsymbol{x}_A \,\right)[P]\left\langle\, \boldsymbol{s}_B \,\right\rangle$, we have $f \otimes U = \left(\, \boldsymbol{x}_A \;\; \boldsymbol{y}_U \,\right)[P]\left\langle\, \boldsymbol{s}_B \;\; \boldsymbol{y}_U \,\right\rangle$ and thus $P^\dagger(\boldsymbol{y}_U/\boldsymbol{y}_U) = P$ because $\boldsymbol{y}_U$ does not occur in P. The final requirement, that regular scalars $()[P]\langle\rangle$ are invertible, follows from the fact that the variables in a closed cord P must be bound by the (νm) operator. The scalar denominator in a morphism $\frac{f}{s} = \frac{(x_A)[P]\langle s_B\rangle}{()[P']\langle\rangle}$ displays the random nonces of the cord P, and the equivalence $\frac{f}{s} \sim \frac{g}{t}$ identifies the processes modulo their fresh nonces. This completes the proof that the iterative structure on $\mathcal{T}$ induces a trace structure on $\mathcal{P} = \mathcal{P}_{\mathcal{T},\mathcal{W},\mathcal{A}}$.

The converse, that the trace operation in $\mathcal{P}$ induces an iteration operation in $\mathcal{T}$ is proven by retracing a suitable special case of the above construction backwards. Given a guarded system $\boldsymbol{y}_{1..k} = \boldsymbol{s}_{1..k}(\boldsymbol{y}_{1..k})$, consider the program $s : 0 + k \longrightarrow k + k$ in the form

$$\left(\, \boldsymbol{y}_{1..k} \,\right)[]\left\langle\, \boldsymbol{y}_{1..k} \;\; \boldsymbol{s}_{1..k}(\boldsymbol{y}_{1..k}) \,\right\rangle$$

and use the properties of the trace to show that the program $Tr^k_{0k}(s)$, which is in the form $()[]\langle \boldsymbol{s}^{\,\dagger}_{1..k}\rangle$, gives the unique solution of the given system. $\square$

Remark. Note that the trace operation generally makes some of the previously bound variables. In particular, the variables $\boldsymbol{y}^*_{k..\ell}$, formed from the repetitions of a subset of $\boldsymbol{y}_{k..\ell}$ that the non-guarded part of the system $\boldsymbol{y} = \boldsymbol{s}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v})$ induces, are free in $Tr(p)$, although the tuple $\boldsymbol{y}_{k..\ell}$ was of course bound in p. This means that the polynomial subcategories $\mathcal{P}[\boldsymbol{v}]$ of $\mathcal{P}$ are generally not closed under the described trace operation, and just support a *partial* trace operation. The semantical significance of this will become clear later.

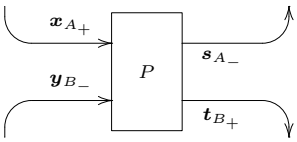
3.2 Interactions

An *interaction* is a cord process whose input and output interface is split into an *Initiator's* part (i.e. the domain), and a *Responder's* part (the codomain).[4]

The category $\mathcal{J}$ of teams is obtained by applying the Int-construction [29] to the category $\mathcal{P}$ of programs. The objects of $\mathcal{J}$ are pairs of arities. Since they will usually correspond to agents, we give them names A, B, C etc., and write $A = \langle A_+, A_- \rangle$ and $B = \langle B_+, B_- \rangle$, where A_+, A_-, B_+, B_- are some arities. An interaction of $A = \langle A_+, A_- \rangle$ and $B = \langle B_+, B_- \rangle$ is described by a morphism $p : A \longrightarrow B$, which is a program $p : A_+ \otimes B_- \longrightarrow A_- \otimes B_+$, usually written in the form

$$\begin{pmatrix} \boldsymbol{x}_{A_+} \\ \boldsymbol{y}_{B_-} \end{pmatrix} \left[\ P\ \right] \left\langle \begin{matrix} \boldsymbol{s}_{A_+} \\ \boldsymbol{t}_{B_-} \end{matrix} \right\rangle = \begin{pmatrix} x_1 & x_2 & \dots & x_{A_+} \\ y_1 & y_2 & \dots & y_{B_-} \end{pmatrix} \left[\ P\ \right] \left\langle \begin{matrix} s_1 & s_2 & \dots & s_{A_-} \\ t_1 & t_2 & \dots & t_{B_+} \end{matrix} \right\rangle$$

or graphically depicted as

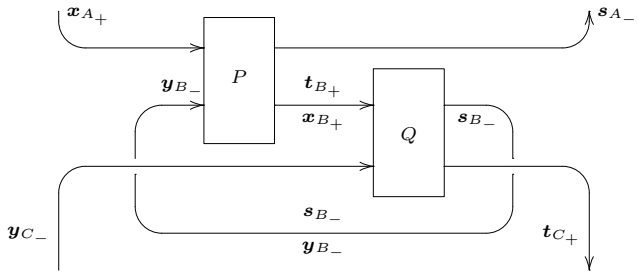


Note that the sign (polarity) of arities changes between the domain and the codomain: at the domain A, the input arity is A_+, and the output is A_-, while the codomain B has the polarities switched, and B_+ is the output arity, while B_- is the input. The point of this is that in the composite $p \circ q : A \longrightarrow C$ of the interactions $p : A \longrightarrow B$ and $q : B \longrightarrow C$, represented by the programs $p : A_+ \otimes B_- \longrightarrow A_- \otimes B_+$ and $q : B_+ \otimes C_- \longrightarrow B_- \otimes C_+$ feeds

- the B_+-outputs $\boldsymbol{t}_B$ of p to the B_+ inputs $\boldsymbol{x}_B$ of q, and
- the B_- outputs $\boldsymbol{s}_B$ of q back to the B_- inputs $\boldsymbol{y}_B$ of p.

[4] The Initiator can be viewed as the System, whereas the Responder as the Environment.

Graphically, the composite $p \circ q : A \longrightarrow C$ is thus



The loop is executed using the trace operation defined in the preceding section. The syntactic view of this operation is:

$$\begin{array}{ccc}
\begin{pmatrix} x_1^A & \ldots & x_{A_+}^A \\ y_1^B & \ldots & y_{B_-}^B \end{pmatrix} & \Big[P\Big] & \left\langle \begin{matrix} s_1^A & \ldots & s_{A_-}^A \\ t_1^B & \ldots & t_{B_+}^B \end{matrix} \right\rangle \\
& \circ & \\
\begin{pmatrix} x_1^B & \ldots & x_{B_+}^B \\ y_1^C & \ldots & y_{C_-}^C \end{pmatrix} & \Big[Q\Big] & \left\langle \begin{matrix} s_1^B & \ldots & s_{B_-}^B \\ t_1^C & \ldots & t_{C_+}^C \end{matrix} \right\rangle \\
& = & \\
\begin{pmatrix} x_1^A & \ldots & x_{A_+}^A \\ y_1^C & \ldots & y_{C_-}^C \end{pmatrix} & \left[\begin{matrix} (P(\boldsymbol{s}_{B_-}^\dagger / \boldsymbol{y}_{B_-}) \otimes C_-) \oslash \\ (A_- \otimes Q(\boldsymbol{t}_{B_+}^\dagger / \boldsymbol{x}_{B_+}) \end{matrix} \right] & \left\langle \begin{matrix} s_1^A & \ldots & s_{A_-}^A(\boldsymbol{s}_{B_-}^\dagger / \boldsymbol{y}_{B_-}) \\ t_1^C & \ldots & t_{C_+}^C(\boldsymbol{t}_{B_+}^\dagger / \boldsymbol{x}_{B_+}) \end{matrix} \right\rangle
\end{array}$$

where $\boldsymbol{s}_{B_-}^\dagger$ and $\boldsymbol{t}_{B_+}^\dagger$ constitute the solution of the system:

$$\begin{aligned}
\boldsymbol{y}_{B_-} &= \boldsymbol{s}_{B_-}\left(\boldsymbol{x}_{B_+}, \boldsymbol{y}_{C_-}\right) \\
\boldsymbol{x}_{B_+} &= \boldsymbol{t}_{B_+}\left(\boldsymbol{x}_{A_+}, \boldsymbol{y}_{B_-}\right)
\end{aligned}$$

The fact that, by the laws of the iterative algebras, these solutions can be extracted in any order, either from

$$\begin{aligned}
\boldsymbol{y}_{B_-} &= \boldsymbol{s}_{B_-}\left(\boldsymbol{t}_{B_+}\left(\boldsymbol{x}_{A_+}, \boldsymbol{y}_{B_-}\right), \boldsymbol{y}_{C_-}\right) \text{ or from} \\
\boldsymbol{x}_{B_+} &= \boldsymbol{t}_{B_+}\left(\boldsymbol{x}_{A_+}, \boldsymbol{s}_{B_-}\left(\boldsymbol{x}_{B_+}, \boldsymbol{y}_{C_-}\right)\right)
\end{aligned}$$

implies that

$$\begin{array}{ccc}
\begin{pmatrix} x_1^A & \ldots & x_{A_+}^A \\ y_1^C & \ldots & y_{C_-}^C \end{pmatrix} & \left[\begin{matrix} (P(\boldsymbol{s}_{B_-}^\dagger / \boldsymbol{y}_{B_-}) \otimes C_-) \oslash \\ (A_- \otimes Q(\boldsymbol{t}_{B_+}^\dagger / \boldsymbol{x}_{B_+}) \end{matrix} \right] & \left\langle \begin{matrix} s_1^A & \ldots & s_{A_-}^A(\boldsymbol{s}_{B_-}^\dagger / \boldsymbol{y}_{B_-}) \\ t_1^C & \ldots & t_{C_+}^C(\boldsymbol{t}_{B_+}^\dagger / \boldsymbol{x}_{B_+}) \end{matrix} \right\rangle \\
& = & \\
\begin{pmatrix} x_1^A & \ldots & x_{A_+}^A \\ y_1^C & \ldots & y_{C_-}^C \end{pmatrix} & \left[\begin{matrix} (A_+ \otimes Q(\boldsymbol{t}_{B_+}^\dagger / \boldsymbol{x}_{B_+}) \oslash \\ (P(\boldsymbol{s}_{B_-}^\dagger / \boldsymbol{y}_{B_-}) \otimes C_+) \end{matrix} \right] & \left\langle \begin{matrix} s_1^A & \ldots & s_{A_-}^A(\boldsymbol{s}_{B_-}^\dagger / \boldsymbol{y}_{B_-}) \\ t_1^C & \ldots & t_{C_+}^C(\boldsymbol{t}_{B_+}^\dagger / \boldsymbol{x}_{B_+}) \end{matrix} \right\rangle
\end{array}$$

which corresponds to the transformation of the above graphic representation, where the Q-box would be moved to the left of the P-box along the loop.

Note that these equivalent forms of the definition are just the unfoldings of the general formulas[5]

$$\begin{aligned} p \circ q &= Tr^{B_-}\Big(A_+ \otimes B_- \otimes C_- \xrightarrow{p\otimes C_-} A_- \otimes B_+ \otimes C_- \xrightarrow{A_-\otimes q} A_- \otimes B_- \otimes C_+\Big) \\ &= Tr^{B_-\otimes B_+}\Big(A_+ \otimes B_- \otimes B_+ \otimes C_- \xrightarrow{p\otimes q} A_- \otimes B_+ \otimes B_- \otimes C_+\Big) \\ &= Tr^{B_+}\Big(A_+ \otimes B_+ \otimes C_- \xrightarrow{A_+\otimes q} A_+ \otimes B_- \otimes C_+ \xrightarrow{p\otimes C_+} A_- \otimes B_+ \otimes C_-\Big) \end{aligned}$$

which equivalently define the composition of $p : A_+ \otimes B_- \longrightarrow A_- \otimes B_+$ and $q : B_+ \otimes C_- \longrightarrow B_- \otimes C_+$ in the free compact category $\mathsf{Int}(\mathbb{C})$ over an arbitrary traced monoidal category $\mathbb{C}$. The compact structure of the category $\mathcal{J}$ of interactions is thus defined as usually in $\mathsf{Int}(\mathbb{C})$, because $\mathcal{J} = \mathsf{Int}(\mathcal{P})$. The unit $\eta : 0 \longrightarrow A^* \otimes A$ and the counit $\varepsilon : A \otimes A^* \longrightarrow 0$ both correspond to the buffer on A.

The monoid of scalars consists of all cord spaces $\mathcal{L}$, taken with both interfaces empty. The scalar multiplication is the parallel composition.[6] The embeddings

$$\begin{array}{rcl} \mathsf{Init} : \mathcal{P} & \longrightarrow & \mathcal{J} \\ n & \longmapsto & \langle n, 0\rangle \end{array} \qquad\qquad \begin{array}{rcl} \mathsf{Resp} : \mathcal{P}^{op} & \longrightarrow & \mathcal{J} \\ n & \longmapsto & \langle 0, n\rangle \end{array}$$

map programs respectively to the Initiator-only interactions and the Responder-only interactions. They both display $\mathcal{J}$ as the free compact category over $\mathcal{P}$.

If semantics for actions in $\mathcal{A}$ is given in such a way that processes, presented as cord spaces, are reversible, with each send consumed by a single receive, and with an involution $\dagger : \mathcal{L} \longrightarrow \mathcal{L}$ inverting the order of actions, then the category $\mathcal{P}$ comes with an involutive functor $\dagger : \mathcal{P}^{op} \longrightarrow \mathcal{P}$. This functor lifts along the Int-construction, and $\mathcal{J}$ becomes a $\dagger$-compact category too, suitable for presenting, and perhaps analyzing quantum protocols [4, 23].

Remark. The category $\mathcal{P}$ of cord processes has all projections $\big(\ x\ y\ \big)\ []\ \big\langle\ x\ \big\rangle$ and diagonals $\big(\ x\ \big)\ []\ \big\langle\ x\ x\ \big\rangle$ (albeit not natural transformations, so $\mathcal{P}$ is not cartesian). Neither of these families is preserved under the $\mathcal{J}$-construction, and $\mathcal{J}$ only has the diagonals and projections for the embedded copies of the Initiator-only and Responder-only programs.

4 Protocols and attacks as interactions

A **protocol** consists of a process and a nonempty set of the desired runs. We present protocol processes as cord processes, and suggest the desired runs typographically: the local time of each cord spaces flows top down, whereas the

[5] For simplicity, we omit the evident commutation isomorphisms.

[6] Note that $s \circ t = Tr^0(s \otimes t)$ gives $s \circ t = s \otimes t$ for the scalars in $\mathcal{J}$, whereas in $\mathcal{P}$ the parallel and the sequential composition of the programs $s, t : 0 \longrightarrow 0$ are quite different.

desired interactions are aligned horizontally. — Note that the local time in cords written as process expressions, like we did so far, flows left to right. We change this convention in this final section, in order to be able to fit a protocol on a page.

When they participate a protocol, the agents[7] play various *roles* in it. Formally, roles can thus be viewed as agent variables, that get instantiated to agent names when a particular agent assumes a role in a particular protocol run [35, 13].

4.1 The Needham-Schroeder Public Key protocol (NSPK)

Prerequisites. To dam the flood of parentheses, we write functions in a curried form: a function of two arguments is written Ekx instead of $E(k,x)$. This leaves $(-,-) : \mathcal{T} \times \mathcal{T} \longrightarrow \mathcal{T}$ to denote the pairing operation. Formally, we assume that the agent identifiers are terms, i.e. $\mathcal{W} \subset \mathcal{T}$, so that any operation on $\mathcal{T}$ can also be applied on $\mathcal{W}$.

An abstract form of the Public Key Infrastructure is expressed by the assumption that all agents given in advance the maps $E, D : \mathcal{T} \longrightarrow \mathcal{T}$ and $k : \mathcal{W} \longrightarrow \mathcal{T}$, which satisfy the equation

$$D\overline{k}_X(Ek_X y) = y \tag{2}$$

for all $X : \mathcal{W}$ and $y : \mathcal{T}$, and for map $\overline{k} : \mathcal{W} \longrightarrow \mathcal{T}$, which is not publicly known. Formally, these three maps are given as the common context to all processes, represented as cords. In other words, we begin from the cord category $\mathbb{P}[E, D, k]$.

The protocol. The cord process representing the NSPK protocol is:

$$\begin{pmatrix} X & \overline{k}_X & Y \\ Y' & \overline{k}_{Y'} & \end{pmatrix} \begin{bmatrix} (\nu m)_X & \\ \langle Ek_Y(X,m)\rangle_X & (u')_{Y'} \\ & \big(X', m' = D\overline{k}_{Y'}u'\big)_{Y'} \\ & (\nu n')_{Y'} \\ (x)_X & \langle Ek_{X'}(m',n')\rangle_{Y'} \\ \big(m, n = D\overline{k}_X x\big)_X & \\ \langle Ek_Y n\rangle_X & (w')_{Y'} \\ & \big(n' = D\overline{k}_{Y'}w'\big)_{Y'} \end{bmatrix} \left\langle \begin{matrix} X & Y & m & n \\ X' & Y' & m' & n' \end{matrix} \right\rangle$$

Note again that the ordering of the cord space is now top-down, rather than left-right; and that the spaces between the actions are introduced to align horizontally the actions that should correspond to each other in the desired run of the protocol. When confusion seems unlikely, we write $action_{Agent}$ instead of a whenever a satisfies $a\ isA\ action_{Agent}$. For simplicity, we omit the source and destination fields from the send and receive actions, and write e.g. $\langle t\rangle$ instead $\langle A \to B : t\rangle$.

[7] Recall from Sec. 2.1 that the terms *agent* and *location* are used in cord calculus interchangeably, denoting the elements of the set $\mathcal{W}$.

Semantics of actions is described in [13, 44]. The send and the receive actions are largely self-explanatory, as is the fresh value generation (νm). Executing a match action $(s_1, \ldots, s_k = t_1, \ldots t_\ell)$, succeeds if $k = \ell$ and for every i such that $1 \leq i \leq k$,

- either s_i is a closed term and $s_i = t_i$,
- or s_i is a variable, and the effect of matching is the assignment $s_i := t_i$.

An obvious security requirement from the NSPK protocol is that for every run of the process represented by the above cord process holds that

- $X = X'$ and $Y = Y'$, i.e. X and Y know that they share the session, and
- $m = m'$ and $n = n'$, i.e. they share the same values.

For the run connecting the sends and the receives written on the same line, this follows from the assumptions that for every X and x

- only X knows $\overline{k}_X$, and3
- the only way to extract x from $Ek_X x$ is via (2).

A stronger security requirement is that upon the completion of a run, the freshly created values m and n are *only* known to X and Y.

The attack. However, besides the desired run, suggested above, the NSPK protocol has other runs. E.g., consider the cord processes NSPK_1 and NSPK_2 in Fig. 1. They are derived from the NSPK by modifying in NSPK_1 the Responder, and in NSPK_2 the Initiator. In both cases, instead of generating a fresh value, the agent takes it from the input interface. Moreover, the challenge received from the peer is forwarded to the output interface. To compose NSPK_1 and NSPK_2, we proceed as in section 3.2

- connect the Responder interfaces of NSPK_1 to the Initiator interfaces of NSPK_2, as suggested by the chosen names
- extend the parallel composition of NSPK_1 and NSPK_2 by the ordering imposed by condition (1), as follows:
  - $X', m' \in \mathrm{BV}\left(X', m' = D\overline{k}_{Z'} u'\right) \cap \mathrm{FV}\left\langle Ek_{Y'}(X', m')\right\rangle$
    $\Longrightarrow \left(X', m' = D\overline{k}_{Z'} u'\right)^1_{Z'} < \left\langle Ek_{Y'}(X', m')\right\rangle^2_{Z'}$
  - $z' \in \mathrm{BV}\left(z'\right) \cap \mathrm{FV}\left\langle Ek_{X'} z'\right\rangle$
    $\Longrightarrow \left(z'\right)^2_{Z'} < \left\langle Ek_{X'} z'\right\rangle^1_{Z'}$,
  - $n' \in \mathrm{BV}\left(n' = D\overline{k}_{Z'} w'\right) \cap \mathrm{FV}\left\langle Ek_{Y'} n'\right\rangle$
    $\Longrightarrow \left(n' = D\overline{k}_{Z'} w'\right)^1_{Z'} < \left\langle Ek_{Y'} n'\right\rangle^2_{Z'}$

where the superscript $(-)^1$ denotes the actions of NSPK_1 and $(-)^2$ the actions of NSPK_2. The resulting interaction $\mathrm{NSPK}_1 \circ \mathrm{NSPK}_2$ is displayed in Fig. 1. Upon the termination of the run of the resulting cord process, $m = m''$ and $n = n''$ will hold, as well as $X = X''$, but $Z \neq Y''$. This means that the requirement that X and Y share the session with each other is not satisfied, since X thinks that she shares it with $Z \neq Y$, whereas Y thinks she shares it with $X = X''$. Moreover, Z knows both freshly generated values m and n, and they are thus not secret between X and Y.

$$\mathrm{NSPK}_1 \quad = \quad \begin{pmatrix} X & \overline{k}_X & Z \\ Z' & \overline{k}_{Z'} & z' \end{pmatrix} \begin{bmatrix} (\nu m)_X & \\ \langle Ek_Z(X,m)\rangle_X & (u')_{Z'} \\ & \left(X', m' = D\overline{k}_{Z'} u'\right)_{Z'} \\ (x)_X & \langle Ek_{X'} z'\rangle_{Z'} \\ \left(m, n = D\overline{k}_X x\right)_X & \\ \langle Ek_Z n\rangle_X & (w')_{Z'} \\ & \left(n' = D\overline{k}_{Z'} w'\right)_{Z'} \end{bmatrix} \left\langle \begin{matrix} X & Z & m & n \\ X' & Z' & m' & n' & \overline{k}_{Z'} & Y' \end{matrix} \right\rangle$$

$$\mathrm{NSPK}_2 \quad = \quad \begin{pmatrix} X' & Z' & m' & n' & \overline{k}_{Z'} & Y' \\ Y'' & \overline{k}_{Y''} & & & & \end{pmatrix} \begin{bmatrix} \langle Ek_{Y'}(X',m')\rangle_{Z'} & (u'')_{Y''} \\ & \left(X'', m'' = D\overline{k}_{Y''} u''\right)_{Y''} \\ & (\nu n'')_{Y''} \\ (z')_{Z'} & \langle Ek_{X''}(m'',n'')\rangle_{Y''} \\ \langle Ek_{Y'} n'\rangle_Z & (w'')_{Y''} \\ & \left(n'' = D\overline{k}_{Y''} w''\right)_{Y''} \end{bmatrix} \left\langle \begin{matrix} Z' & \overline{k}_{Z'} & z' \\ X" & Y'' & m'' & n'' \end{matrix} \right\rangle$$

$$\mathrm{NSPK}_{\mathbf{1}} \circ \mathrm{NSPK}_{\mathbf{2}} \quad = \quad \begin{pmatrix} X & \overline{k}_X & Z \\ Y'' & \overline{k}_{Y''} & \end{pmatrix} \begin{bmatrix} (\nu m)_X & & \\ \langle Ek_Z(X,m)\rangle_X & (u')_{Z'} & \\ & \left(X', m' = D\overline{k}_{Z'} u'\right)_{Z'} & \\ & \langle Ek_{Y'}(X',m')\rangle_{Z'} & (u'')_{Y''} \\ & & \left(X'', m'' = D\overline{k}_{Y''} u''\right)_{Y''} \\ & & (\nu n'')_{Z''} \\ & (z')_{Z'} & \langle Ek_{X''}(m'',n'')\rangle_{Y''} \\ (x)_X & \langle Ek_{X'} z'\rangle_{Z'} & \\ \left(m, n = D\overline{k}_X x\right)_X & & \\ \langle Ek_Z n\rangle_X & (w')_{Z'} & \\ & \left(n' = D\overline{k}_{Z'} w'\right)_{Z'} & \\ & \langle Ek_{Y'} n'\rangle_{Z'} & (w'')_{Y''} \\ & & \left(n'' = D\overline{k}_{Y''} w''\right)_{Y''} \end{bmatrix} \left\langle \begin{matrix} X & Z & m & n \\ X'' & Y'' & m'' & n'' \end{matrix} \right\rangle$$

Fig. 1. The attack components and their composition

5 Discussion and future work

We provided a general categorical view of the MM attacks, and instantiated it on the NSPK protocol. Although the trace structure and the coalgebraic nature of the MM interactions has been displayed, this categorical view did not turn out to be as simple or as succinct as one would like. This may be due to the cord calculus infrastructure, which was originally designed for use in a software tool [10], and later adapted for human consumption. While a certain amount of verbosity may be unavoidable in security formalisms, there is hope that the incremental approach will lead to more convenient languages [45]. The shortcomings of the underlying process calculus notwithstanding, the presented categorical constructions seem to substantiate the idea that *hiding*, inherent in MM, can be captured using the monoidal trace structure. The most important technical features of the presented categorical analysis seem to be that

- the data flows resulting from the interactions correspond to the *iterative structure*, which resolves the systems of equations induced by the interactions, and thus effectively propagates the terms sent in messages;
- the functorial view of the *normal* traces over the cord category $Tr : \mathcal{P}^{\circlearrowleft} \longrightarrow \mathcal{P}$, arising from this iterative structure, can be used to analyze the possible MM attacks, since all hidden interactions that result in a process f observable in $\mathcal{P}$ can be found in its inverse image $Tr^{-1}(f)$ in $\mathcal{P}^{\circlearrowleft}$.

As intriguing as they may be, both these features are clearly beyond the scope of the present paper (even with its swollen Appendices). If the approach turns out to be effective, the intended next step, as mentioned in the Introduction, would be to try to formalize *chosen protocol attacks* [30]. To add more intrigue to the story, this goal seems to require two monoidal structures, to allow distinguishing the situation

- $A \otimes B$, where the roles A and B are played by the same principal, who controls and can mix all information sent and received in both roles; from the situation
- $A \oplus B$, where the roles A and B only exchange information through messages.

Interestingly, the trace structure, at least in its normal flavor, does seem to have a natural generalization in such a framework, as well as a corresponding Int-construction, capturing the MM-interactions that naturally evolve there.

Acknowledgement. This work was started some 10 years ago, as a collaboration with Samson Abramsky, who had introduced me to semantics of interaction a bit earlier, and to computer science just before that. His influence on the presented ideas cannot be overestimated. On the other hand, as our interactions extended not only across the broad areas of common interest, but also across the great distances that separated us, all the shortcomings of the presented text remain entirely my responsibility. Cord calculus was developed in joint work Matthias Anlauff, Iliano Cervesato, Anupam Datta, Ante Derek, Nancy Durgin, Cathy Meadows, and John Mitchell.

References

1. S. Abramsky. Semantics of interaction: an introduction to game semantics. In P. Dybjer and A. Pitts, editors, *Proceedings of the 1996 CLiCS Summer School, Isaac Newton Institute*, pages 1–31. Cambridge University Press, 1997.
2. Samson Abramsky. Interaction categories. In Geoffrey L. Burn, Simon J. Gay, and Mark Ryan, editors, *Theory and Formal Methods*, Workshops in Computing, pages 57–69. Springer, 1993.
3. Samson Abramsky. Algorithmic game semantics: A tutorial introduction. In H. Schwichtenberg and R. Steinbrüggen, editors, *Proceedings of the NATO Advanced Study Institute, Marktoberdorf*, pages 21–47. Kluwer Academic Publishers, 2001.
4. Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 2004. Also arXiv:quant-ph/0402130.
5. Samson Abramsky and Radha Jagadeesan. New foundations for the geometry of interaction. *Information and Computation*, 111(1):53–119, 1994.
6. Peter Aczel, Jiri Adamek, Stefan Milius, and Jiri Velebil. Infinite trees and completely iterative theories: a coalgebraic view. *Theor. Comput. Sci.*, 300(1-3):1–45, 2003.
7. Jiri Adamek. Introduction to coalgebra. *Theory and Applications of Categories*, 14:157–199, 2005.
8. Jiri Adamek, Stefan Milius, and Jiri Velebil. Free iterative theories: a coalgebraic view. *Mathematical. Structures in Comp. Sci.*, 13(2):259–320, 2003.
9. Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.
10. Matthias Anlauff, Dusko Pavlovic, Richard Waldinger, and Stephen Westfold. Proving authentication properties in the Protocol Derivation Assistant. In Pierpaolo Degano, Ralph Küsters, and Luca Vigano, editors, *Proceedings of FCS-ARSPA 2006*. ACM, 2006.
11. Stephen L. Bloom and Calvin C. Elgot. The existence and construction of free iterative theories. *J. Comput. Syst. Sci.*, 12(3):305–318, 1976.
12. Stephen L. Bloom and Zoltan Esik. *Iteration theories: the equational logic of iterative processes*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.
13. Iliano Cervesato, Catherine Meadows, and Dusko Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In Joshua Guttman, editor, *Proceedings of CSFW 2005*, pages 48–61. IEEE, 2005.
14. Anupam Datta, Ante Derek, John Mitchell, and Dusko Pavlovic. Secure protocol composition. *E. Notes in Theor. Comp. Sci.*, pages 87–114, 2003.
15. Anupam Datta, Ante Derek, John Mitchell, and Dusko Pavlovic. A derivation system and compositional logic for security protocols. *J. of Comp. Security*, 13:423–482, 2005.
16. Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system for security protocols and its logical formalization. In Dennis Volpano, editor, *Proceedings of CSFW 2003*, pages 109–125. IEEE, 2003.
17. Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. Secure protocol composition (extended abstract). In Michael Backes, David Basin, and Michael Waidner, editors, *Proceedings of FMCS 2003*, pages 11–23. ACM, 2003.
18. Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. Abstraction and refinement in protocol derivation. In Riccardo Focardi, editor, *Proceedings of CSFW 2004*, pages 30–47. IEEE, 2004.

19. Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Des. Codes Cryptography*, 2(2):107–125, 1992.
20. Cory Doctorow. Solving and creating captchas with free porn. `boingboing.net/2004/01/27/solving-and-creating.html`, retrieved on 2012/1/2.
21. Nancy Durgin, John Mitchell, and Dusko Pavlovic. A compositional logic for proving security properties of protocols. *J. of Comp. Security*, 11(4):677–721, 2004.
22. Nancy Durgin, John C. Mitchell, and Dusko Pavlovic. A compositional logic for protocol correctness. In Steve Schneider, editor, *Proceedings of CSFW 2001*, pages 241–255. IEEE, 2001.
23. Dusko Pavlovic. Geometry of abstraction in quantum computation. In Michael Mislove and Samson Abramsky, editors, *Clifford Lectures 2008*, Proceedings of Symposia in Applied Mathematics. AMS, 2012. 28 pp, arxiv.org:1006.1010.
24. Javier Thayer Fabrega, Jonathan Herzog, and Joshua Guttman. Strand spaces: Proving security protocols correct. *J. Comp. Security*, 7(2/3):191–230, 1999.
25. Jean-Yves Girard. Towards a geometry of interaction. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 69–108. American Mathematical Society, 1989.
26. Jay L. Gischer. The equational theory of pomsets. *Theor. Comp. Sci.*, 61(2-3):199–224, November 1988.
27. Masahito Hasegawa. The uniformity principle on traced monoidal categories. *Electr. Notes Theor. Comput. Sci.*, 69:137–155, 2002.
28. J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.
29. Andre Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.
30. John Kelsey, Bruce Schneier, and David Wagner. Protocol interactions and the chosen protocol attack. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 1997.
31. Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.
32. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.
33. Bill Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences of the United States of America*, 50(1):869–872, 1963.
34. Saunders Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer-Verlag, 1971. (second edition 1997).
35. Catherine Meadows and Dusko Pavlovic. Deriving, attacking and defending the GDOI protocol. In Peter Ryan, Pierangela Samarati, Dieter Gollmann, and Refik Molva, editors, *Proceedings of ESORICS 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 53–72. Springer Verlag, 2004.
36. Catherine Meadows, Radha Poovendran, Dusko Pavlovic, LiWu Chang, and Paul Syverson. Distance bounding protocols: authentication logic analysis and collusion attacks. In R. Poovendran, C. Wang, and S. Roy, editors, *Secure Localization and Time Synchronization in Wireless Ad Hoc and Sensor Networks*. Springer Verlag, 2006.
37. Robin Milner. Action calculi, or syntactic action structures. In *MFCS '93: Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science*, pages 105–121, London, UK, 1993. Springer-Verlag.

38. Robin Milner. Calculi for interaction. *Acta Informatica*, 33(8):707–737, 1996.
39. Lawrence S. Moss. Parametric corecursion. *Theor. Comp. Sci.*, 260(1-2):139–163, 2001.
40. Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21:993–999, December 1978.
41. Dusko Pavlovic. Categorical logic of names and abstraction in action calculus. *Math. Structures in Comp. Sci.*, 7:619–637, 1997.
42. Dusko Pavlovic. Network as a computer: ranking paths to find flows. In Alexander Razborov and Anatol Slissenko, editors, *Proceedings of CSR 2008*, volume 5010 of *Lecture Notes in Computer Science*, pages 384–397. Springer Verlag, 2008. arxiv.org:0802.1306.
43. Dusko Pavlovic and Samson Abramsky. Specifying interaction categories. In E. Moggi and G. Rosolini, editors, *Category Theory and Computer Science '97*, volume 1290 of *Lecture Notes in Computer Science*, pages 147–158. Springer Verlag, 1997.
44. Dusko Pavlovic and Catherine Meadows. Deriving secrecy properties in key establishment protocols. In Dieter Gollmann and Andrei Sabelfeld, editors, *Proceedings of ESORICS 2006*, volume 4189 of *Lecture Notes in Computer Science*. Springer Verlag, 2006.
45. Dusko Pavlovic and Catherine Meadows. Actor Network Procedures. In Ram Ramanujam and Srini Ramaswamy, editors, *Proceedings of International Conference on Distributed Computing and Internet Technologies 2012*, volume 7154 of *Lecture Notes in Computer Science*, page 20 pp. Springer Verlag, 2012.
46. Vaughan Pratt. Modelling concurrency with partial orders. *Internat. J. Parallel Programming*, 15:33–71, 1987.
47. Ronald L. Rivest and Adi Shamir. How to expose an eavesdropper. *Commun. ACM*, 27:393–394, April 1984.

A Appendix: Traces over local monoidal categories

For simplicity, and without loss of generality, we assume that the monoidal structures that we consider are strictly associative and unitary, i.e. $(A \otimes B) \otimes C = A \otimes (B \otimes C)$ and $A \otimes I = I \otimes A = A$.

A.1 Local monoidal categories and loop categories

Definition A.1. *A small symmetric monoidal category*

$$\mathbb{C} \times \mathbb{C} \xrightarrow{\otimes} \mathbb{C} \xleftarrow{I} 1$$

is said to be graded *by a monoid homomorphism*

$$(\mathbb{C}, \otimes, I) \xrightarrow{|-|} (\mathbb{I}, \circ, 1)$$

where $\mathbb{I} = \mathbb{C}(I, I)$, $1 = \mathrm{id}_I$. *The elements of*

$$\mathbb{I}^* = \{s \in \mathbb{I} \mid \forall n \in \mathbb{N} \exists u \in \mathbb{I}.\ s^n u \neq s^n\}$$

are called regular. *An object U is regular whenever its grade $|U|$ is regular. A graded symmetric monoidal category is called* local monoidal *if all of its regular scalars are invertible.*

Definition A.2. *For a local monoidal category $\mathbb{C}$, the trace structure in the sense of [29] is said to be* normal *if, in addition to the standard axioms the trace operation also satisfies the* normality *requirement:*

$$Tr^U\left(A \otimes U \xrightarrow{f \otimes U} B \otimes U\right) = \left(A \xrightarrow{f} B\right)$$

for every regular U.

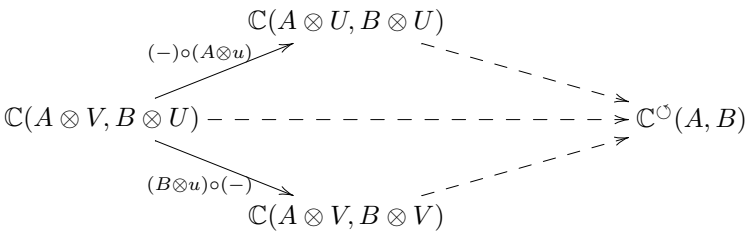
Remark. The trace structures with respect to both the additive and the multiplicative monoidal structure of the category relations are normal. On the other hand, the standard trace structure over the category of vector spaces is not normal. The upshot of the normality requirement is that it opens a functorial view of the traces.

Loop category. Given a local monoical category $\mathbb{C}$, let $\mathbb{C}^{\circlearrowleft}$ be the category defined

$$\begin{aligned} |\mathbb{C}^{\circlearrowleft}| &= |\mathbb{C}| \\ \mathbb{C}^{\circlearrowleft}(A,B) &= \Big(\sum_{U \in |\mathbb{C}|} \mathbb{C}(A \otimes U, B \otimes U) \ \times \ \mathbb{I}^*\Big) \Big/ \sim \end{aligned}$$

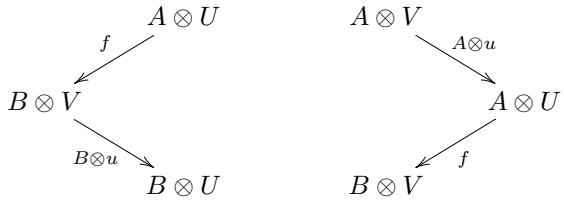
A $\mathbb{C}^{\circlearrowleft}$-morphism from A to B is thus an equivalence class of pairs $\langle f, s\rangle$, where $f : A \otimes U \longrightarrow B \otimes U$ is a $\mathbb{C}$-morphism and $s : I \longrightarrow I$ is a regular scalar. Writing such pairs as fractions $\frac{f}{s}$, we define $\sim$ as the smallest equivalence relation containing the following relations

– the coend equivalence [34, IX.6]



which means

$$\frac{(B \otimes u) \circ f}{s} \sim \frac{f \circ (A \otimes u)}{s}$$



– tensor is normalized

$$\frac{f \otimes U}{s \circ |U|} \quad \sim \quad \frac{f}{s}$$

$$\begin{array}{cc} A \otimes U & A \\ {\scriptstyle f\otimes U}\downarrow & \downarrow{\scriptstyle f} \\ B \otimes U & B \end{array}$$

– regular scalars are invertible in all morphisms

$$\frac{f}{s} \sim \frac{g}{t} \iff \exists uv \in \mathbb{I}^*.\ \ u \circ f = v \circ g \wedge u \circ s = v \circ t$$

We proceed to define the categorical structure of $\mathbb{C}^{\circlearrowleft}$. Given

– $f \in \mathbb{C}^{\circlearrowleft}(A, B)$ as $A \otimes U \xrightarrow{f_0/f_1} B \otimes U$,
– $g \in \mathbb{C}^{\circlearrowleft}(B, C)$ as $B \otimes V \xrightarrow{g_0/g_1} C \otimes V$, and
– $h \in \mathbb{C}^{\circlearrowleft}(C, D)$ as $C \otimes V \xrightarrow{h_0/h_1} D \otimes V$

then the **composition** $f \circ g \in \mathbb{C}^{\circlearrowleft}(A, C)$ can be viewed as

$$\begin{array}{ccc} A \otimes U \otimes V & \xrightarrow[\frac{f_0\otimes V}{f_1 \circ |V|}]{} & B \otimes U \otimes V \\ & & {\scriptstyle B\otimes c}\downarrow \\ & & B \otimes V \otimes U \end{array} \qquad \begin{array}{ccc} C \otimes U \otimes V \\ {\scriptstyle C\otimes c}\uparrow \\ C \otimes V \otimes U \end{array}$$

$$B \otimes V \otimes U \xrightarrow[\frac{g_0\otimes U}{g_1 \circ |U|}]{} C \otimes V \otimes U$$

or equivalently

$$\begin{array}{ccc} A \otimes U \otimes V & \xrightarrow[\frac{f_0\otimes V}{f_1 \circ |V|}]{} & B \otimes U \otimes V \\ {\scriptstyle A\otimes c}\uparrow & & {\scriptstyle B\otimes c}\downarrow \\ A \otimes V \otimes U & & B \otimes V \otimes U \end{array} \xrightarrow[\frac{g_0\otimes U}{g_1 \circ |U|}]{} C \otimes V \otimes U$$

whereas the **tensor** $f \otimes h \in \mathbb{C}^{\circlearrowleft}(A \otimes C, B \otimes D)$ is

$$\begin{array}{ccc} A \otimes C \otimes U \otimes V & & B \otimes D \otimes U \otimes V \\ {\scriptstyle A\otimes c\otimes V}\downarrow & & {\scriptstyle B\otimes c\otimes V}\uparrow \\ A \otimes U \otimes C \otimes V & \xrightarrow[\frac{f_0\otimes h_0}{f_1 \circ h_1}]{} & B \otimes U \otimes D \otimes V \end{array}$$

Since the scalars in $\mathbb{C}^{\circlearrowleft}$ are the fractions of those in $\mathbb{C}$, the **grading** of $\mathbb{C}^{\circlearrowleft}$ is inherited from $\mathbb{C}$. Finally, the **trace** operation is

$$\frac{f \ = \ \dfrac{(A \otimes U) \otimes V \xrightarrow{f_0} (B \otimes U) \otimes V}{f_1} \ \in \ \mathbb{C}^{\circlearrowleft}(A \otimes U, B \otimes U)}{Tr^U_{AB} f \ = \ \dfrac{A \otimes (U \otimes V) \xrightarrow{f_0} B \otimes (U \otimes V)}{f_1} \ \in \ \mathbb{C}^{\circlearrowleft}(A, B)}$$

To see that the operators $Tr^U_{AB} : \mathbb{C}^{\circlearrowleft}(A \otimes U, B \otimes U) \longrightarrow \mathbb{C}^{\circlearrowleft}(A, B)$ satisfy the trace axioms from [29], observe that

- dinaturality (sliding) and yanking laws are imposed by the definition of $\sim$
- naturality (tightening) by the definition of the composition in $\mathbb{C}^{\circlearrowleft}$, whereas
- vanishing and superposition are easily checked by inspection.

Theorem A.1. *The loop category $\mathbb{C}^{\circlearrowleft}$ is the free normal traced category generated by the local monoidal category $\mathbb{C}$. Normal traced categories correspond to strong algebras $T : \mathbb{C}^{\circlearrowleft} \longrightarrow \mathbb{C}$ for the monad $\circlearrowleft$: $\mathcal{LM} \longrightarrow \mathcal{LM}$ on the category $\mathcal{LM}$ of small local monoidal categories with the grade preserving monoidal functors. The monad structure is*

$$\begin{array}{rcl} \boldsymbol{\eta}_{\mathbb{C}} & : & \mathbb{C} \longrightarrow \mathbb{C}^{\circlearrowleft} \\ (A \xrightarrow{f} B) & \longmapsto & \Big[A \otimes I \xrightarrow{f \otimes I} B \otimes I\Big]_{\sim} \end{array}$$

$$\begin{array}{rcl} \boldsymbol{\mu}_{\mathbb{C}} & : & \mathbb{C}^{\circlearrowleft\circlearrowleft} \longrightarrow \mathbb{C}^{\circlearrowleft} \\ \left[\dfrac{\left[\dfrac{(A\otimes U)\otimes V \xrightarrow{f_0} (B\otimes U)\otimes V}{s}\right]_{\sim}}{t}\right]_{\sim} & \longmapsto & \left[\dfrac{A \otimes (U \otimes V) \xrightarrow{f_0} B \otimes (U \otimes V)}{s \circ t}\right]_{\sim} \end{array}$$

Towards the **proof**, observe that the arrow part of a loop algebra $T : \mathbb{C}^{\circlearrowleft} \longrightarrow \mathbb{C}$ yields a map

$$\sum_{U \in \mathbb{C}} \mathbb{C}(A \otimes U, B \otimes U) \longrightarrow\!\!\!\!\!\rightarrow \mathbb{C}^{\circlearrowleft}(A, B) \xrightarrow{T_{AB}} \mathbb{C}(A, B) \tag{3}$$

which boils down to a family of trace operators

$$\left\{Tr^U_{AB} : \mathbb{C}(A \otimes U, B \otimes U) \longrightarrow \mathbb{C}(A, B)\right\}_{U \in \mathbb{C}}$$

Note that this is not a natural family, since precomposing on the left with $g \otimes U$ corresponds on the right to $g \circ |U|$. The operators $Tr^U_{AB} : \mathbb{C}(A \otimes U, B \otimes U) \longrightarrow \mathbb{C}(A, B)$ do satisfy the trace axioms of [29] because:

- naturalities, yanking, normality $\iff$ factoring through $\mathbb{C}^{\circlearrowleft}(A, B)$,
- superposition $\iff$ $T \circ \boldsymbol{\eta}_{\mathbb{C}} = id_{\mathbb{C}}$
- vanishing $\iff$ $T \circ \boldsymbol{\mu}_{\mathbb{C}} = T \circ T^{\circlearrowleft}$

Remark. Although $\circlearrowleft$ is not a KZ-monad, its restriction to symmetric monoidal posets, i.e. to ordered abelian monoids, is an idempotent monad. An ordered abelian monoid has a trace if and only if the monoid operation is an order isomorphism, i.e. $\exists x.a + x = b + x \Longrightarrow a = b$. The Int-construction generates the ordered abelian groups, since $0 \leq a + a^*$ and $a^* + a \leq 0$ mean that $a^* = -a$.

B Appendix: Uniform traces over local monoidal categories

Definition B.1. *[27] A trace operator is* uniform *if*

$$Tr^U_{AB}(f) = Tr^V_{AB}(g)$$

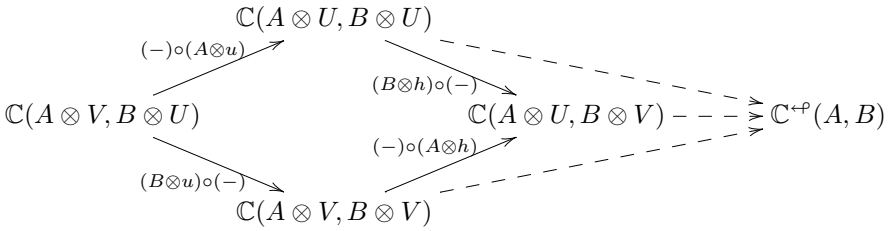
holds whenever some h makes the following diagram commute.

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

Uniform loop category. Given a local monoidal category $\mathbb{C}$, let $\mathbb{C}^{\circlearrowleft}$ be the category defined

$$\begin{aligned} |\mathbb{C}^{\leftarrow\rho}| &= |\mathbb{C}| \\ \mathbb{C}^{\leftarrow\rho}(A,B) &= \Big(\sum_{U\in|\mathbb{C}|} \mathbb{C}(A\otimes U, B\otimes U)\times \mathbb{I}^*\Big) \Big/ \approx \end{aligned}$$

where $\approx$ extends $\sim$ by the following extension of the coend equivalence:



i.e. by adding

$$\begin{array}{ccc} A\otimes U & \xrightarrow{-A\otimes h-} & A\otimes V \\ {\scriptstyle f}\downarrow & \approx & \downarrow{\scriptstyle g} \\ B\otimes U & \xrightarrow[-B\otimes h-]{} & B\otimes V \end{array}$$

Theorem B.1. *The uniform loop category $\mathbb{C}^{\leftarrow\rho}$ is the free normal uniformly traced category generated by the local monoidal category $\mathbb{C}$. Normal uniformly traced categories correspond to the strong algebras $T : \mathbb{C}^{\leftarrow\rho} \longrightarrow \mathbb{C}$ for the monad $\leftarrow\rho\colon \mathcal{GM} \longrightarrow \mathcal{GM}$ on the category $\mathcal{GM}$ of small local monoidal categories with the grade preserving monoidal functors.*

C Appendix: Traced clones

The monadic view of normal traced categories allows effective calculations of the trace structures. For instance, consider the monoid of natural numbers

$$\mathbb{N} \times \mathbb{N} \xrightarrow{+} \mathbb{N} \xleftarrow{0} 1$$

as the category of sets $n = \{0, 1, \ldots, n-1\}$ and functions between them. The grading is trivial. Then

$$\mathbb{N}^{\circlearrowleft}(a,b) = \sum_{u \in \mathbb{N}} \left\{ a + u \xrightarrow{f} b + u \mid \forall y \in u \exists x.\ f(x) = y \ \wedge\ (x \in u \ \vee f(y) \in u) \right\}$$

$$\mathbb{N}^{\leftrightarrow\rho}(a,b) = \sum_{u \in \mathbb{N}} \left\{ a + u \xrightarrow{f} b + u \ \in \mathbb{N}^{\circlearrowleft}(a,b) \mid \forall y \in u.\ f(y) = y \ \vee\ \exists i.\ f^i(y) \in b \right\}$$

These constructions extend to the situations when $\mathbb{N}$ is extended by algebraic operations and actions.

Definition C.1. *[33] Given an algebraic theory $\mathcal{T} = \langle \Sigma_{\mathcal{T}}, E_{\mathcal{T}} \rangle$, where $\Sigma = \Sigma_{\mathcal{T}}$ is a signature, and $E = E_{\mathcal{T}}$ is a set of equations, the induced* clone $\mathbb{N}_{\mathcal{T}} = \mathbb{N}[\Sigma; E]$[8] *is the category*

$$|\mathbb{N}_{\mathcal{T}}| = |\mathbb{N}|$$
$$\mathbb{N}_{\mathcal{T}}(m,n) = \left\{\ (x_1, \ldots, x_n)\langle \varphi_1, \ldots, \varphi_m \rangle\ \right\} / \alpha$$

i.e. obtained by

- *adjoining to $\mathbb{N}$ an arrow $m \xrightarrow{\varphi} n$ for every m-tuple $\langle \varphi_i(x_1, \ldots x_n) \rangle_{i \le m}$ of well-formed Σ-operations, modulo the α-conversion, i.e. variable renaming; and then by*
- *imposing the equations of E on the obtained category.*

Definition C.2. *An algebraic theory $\mathcal{T}$ is* iterative *if every system*

$$\begin{aligned} y_1 &= f_1(y_1, y_2, \ldots, y_k, \ldots, y_\ell) \\ y_2 &= f_2(y_1, y_2, \ldots, y_k, \ldots, y_\ell) \\ &\ldots \\ y_k &= f_k(y_1, y_2, \ldots, y_k, \ldots, y_\ell) \end{aligned}$$

has a unique solution

$$\begin{aligned} f_1^\dagger(y_{k+1}, \ldots, y_\ell) &= f_1(f_1^\dagger, f_2^\dagger, \ldots, f_k^\dagger, \ldots, y_\ell) \\ f_2^\dagger(y_{k+1}, \ldots, y_\ell) &= f_2(f_1^\dagger, f_2^\dagger, \ldots, f_k^\dagger, \ldots, y_\ell) \\ &\ldots \\ f_k^\dagger(y_{k+1}, \ldots, y_\ell) &= f_k(f_1^\dagger, f_2^\dagger, \ldots, f_k^\dagger, \ldots, y_\ell) \end{aligned}$$

provided that all equations are guarded*, i.e. that none of the operations f_j is a projection.*

[8] The notation echoes [41], where free constructions over monoidal categories were analyzed as polynomial extensions.

Theorem C.1. *The uniform traces over the clone $\mathbb{N}_{\mathcal{T}}$ are in one to one correspondence with the iterative structures over the algebraic theory $\mathcal{T}$.*

This is where Prop. 3.1 picks up the thread, with

$$|\mathcal{P}_{\mathcal{T},\mathcal{W},\mathcal{A}}| = |\mathbb{N}|^2$$
$$\mathcal{P}_{\mathcal{T},\mathcal{W},\mathcal{A}}\left(\langle k,m\rangle,\langle \ell,n\rangle\right) = \left\{\ (\boldsymbol{X}_{1..k},\boldsymbol{x}_{1..n})[P]\langle \boldsymbol{A}_{1..\ell},\boldsymbol{\varphi}_{1..m}\rangle\ \right\}/\alpha$$

where $A_1,\ldots,A_\ell \in \mathcal{A}$, $\varphi_1,\ldots,\varphi_m \in \mathcal{T}$ and P are the processes built from the actions in $\mathcal{A}$ over the locations in $\mathcal{W}$.