HAL
open science

Synchronous Machines: a Traced Category

Marc Bagnol, Guatto Adrien

▶ **To cite this version:**

Marc Bagnol, Guatto Adrien. Synchronous Machines: a Traced Category. [Research Report] 2012. hal-00748010v2

HAL Id: hal-00748010

https://inria.hal.science/hal-00748010v2

Submitted on 27 Nov 2012 (v2), last revised 27 Nov 2012 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Synchronous Machines: a Traced Category

Marc Bagnol* & Adrien Guatto†

Institut de Mathématiques de Luminy, Aix-Marseille Université
Département d'Informatique de l'École normale supérieure and INRIA Rocquencourt

Abstract. Synchronous programming languages have been extensively used in the area of critical embedded systems. Synchronous machines, a specific class of labelled transition systems, are often used to give denotational semantics of these languages. In this work, we study the categorical structure of the aforementioned machines.
We first show that the category **S** of synchronous machines can be given a traced symmetric monoidal structure with diagonals.
Then, we apply a standard variant of the **Int** construction to **S** and relate the composition in the resulting category with the synchronous product, the operation used to model parallel composition of synchronous programs. We also show how properties of synchronous machines like determinism and reactivity relate to the way they compose with diagonal morphisms of **S**.

Keywords:
Denotational semantics; Traced categories; Synchronous languages.

1 Introduction

Synchronous programming languages [18] are domain-specific languages dedicated to the design and implementation of critical real-time systems. Based on a strong yet abstract notion of time, they have met success and industrial use. Compared to most practically used programming languages, their semantics tend to be better understood and more formal.

In both computer science and logics, the denotational semantics approach has helped in clarifying the structure of programming languages and logical systems, by means of mathematical tools such as category theory. For instance linear logics [3] comes from the study of a fine-grained denotational model of system F [19].

The traced monoidal categories [20] have been studied in the area of denotational semantics to model "reasonable" *feedback* situations. The $\mathcal{G}$ construction [17] allows to build out of a traced monoidal category a new category in which the composition can be thinked of intuitively as a kind of parallel feedback loop between the composed morphisms.

* This author was supported by the *Agence Nationale de la Recherche* project ANR-10-BLAN-0213

† This author was funded by the European FP7 project PHARAON id. 288307.

Automata theory forms the backbone of synchronous programming. More specifically, parallel composition mirrors the synchronous product of automata: two interacting processes share the same notion of logical time and must synchronize their reaction steps. This operation also provides a natural notion of mutual feedback between concurrent processes, which comes in handy for programming real-time systems. For example, one may use it to implement the interaction between the model of a physical environment and its software controller.

It is then quite natural to wonder if the automata-based models of synchronous programming languages satisfy the axioms of traced monoidal categories. Even if these models were introduced without categories in mind, their parallel composition operator should be related somehow to the parallel composition introduced by the $\mathcal{G}$ construction.

In this work, we first show that the *synchronous machines*, the kind of automata used in these models, form a traced monoidal category $\mathbf{S}$ with diagonals. Then, we show that the parallel composition in $\mathcal{G}(\mathbf{S})$ correspond indeed to the synchronous product. Moreover we study the *diagonal* structure of $\mathbf{S}$, which is a categorical way to speak of duplicability. We show in particular that the class of copyable morphisms of $\mathbf{S}$ corresponds exactly to the one-state deterministic synchronous machines.

2 Synchronous machines

2.1 Context

Pnueli [15] proposed to consider two broad classes of computing systems.

- *Transformational systems*, whose role is to compute an output from an input and then stop. Compilers, automated theorem provers, or ray-tracers are transformational programs.
- *Reactive systems*, repetitively interacting with an external environment. All the software and hardware systems from the field of *real-time systems* fall under the umbrella of reactive systems; let us cite automotive and avionics control software, but also on-line video encoding and image filtering, etc.

Most industrial computing systems are reactive ones. It is therefore not surprising that the design and implementation of programming languages and specification logics tailored to this setting has attracted much interest.

One particular strand of reactive formalisms have met widespread industrial use: the family of synchronous programming languages, such as Esterel [1], Lustre [14] and Signal [10]. The main characteristics of these languages is the notion of a global logical time shared between all processes, with instantaneous communications. An aging but useful survey of the whole area is [18].

Compared to common programming languages and systems for real-time programming, concurrency in synchronous languages is built-in rather than tacked on: it is a linguistic notion. It is also deterministic, in contrast with most works in asynchronous process algebra.

The semantics underlying synchronous languages is rooted in automata-theoretic notions. In such languages, parallel composition corresponds to the synchronous product of automata. This choice is conceptually simple and fits well with the culture of their main users, control and signal scientists.

It is also very relevant for formal verification, a vital activity in critical system design. Indeed, the synchronous product typically results in much smaller systems that the asynchronous one and contributes to keep the state-space explosion problem under control. Moreover, safety properties can themselves be expressed directly through so-called *synchronous observers* [16]. It is then possible to verify, test or simulate the parallel composition of a program and its observers as "syntactic" parallel composition and intersection of execution traces coincide. Observers are synchronous programs and can thus be themselves verified and tested using the same methods.

In this work, we handle the general setting of synchronous products of composable finite automata, following the definitions of [16]. Our goal is then to extract the categorical structure of these objects, and recreate the seemingly primitive notion of synchronous interaction from algebraic means.

2.2 Synchronous machines

We study an abstracted model of synchronous communication rather than a concrete synchronous language. As in [16], we use a variant of finite automata with the synchronous product. However, here we do not suppose that signals come from a predefined global set. This is essential in order to comply with the categorical point of view; we discuss this point further in section 5.

Definition 1. ***Synchronous machines***
A ***synchronous machine*** f *is a 5-tuple* $(A, B, \mathbf{F}, \mathbf{i}_f, \Delta_f)$ *where:*

- $\mathbf{F}$ *is a set of states;*
- $\mathbf{i}_f \in \mathbf{F}$ *is the initial state of* f*;*
- A, B *are sets of input and output signals (respectively);*
- $\Delta_f \subseteq \mathbf{F} \times \mathcal{P}(A) \times \mathcal{P}(B) \times \mathbf{F}$ *is the transition relation, expressing how the machine in a given state reacts to a set of input signals by emitting a set of output signals and moving to another state.*

When f can be deduced from the context, we write $\mathbf{q} \xrightarrow[o]{i} \mathbf{q}'$ for $(\mathbf{q}, i, o, \mathbf{q}') \in \Delta_f$. We also define $\delta_f : \mathbf{F} \times \mathcal{P}(A) \to \mathcal{P}(\mathcal{P}(B) \times \mathbf{F})$ the reaction function of machine f by $\delta_f(\mathbf{q}, i) = \{(o, \mathbf{q}') \mid \exists o, \mathbf{q}'.(\mathbf{q}, i, o, \mathbf{q}') \in \Delta_f\}$.

Properties of synchronous machines

Next we describe properties of synchronous machines. Our categorical presentation will characterize them through purely algebraic means. Note that the two first properties are very desirable for practical purposes.

Definition 2. ***Deterministic machine***
A synchronous machine f is ***deterministic*** *if it has at most one possible reaction for each input in each state:* $\forall\, \mathbf{q}, i\,.\ \mid \delta_f(\mathbf{q}, i) \mid \leq 1$

Definition 3. ***Reactive machine***
A synchronous machine f is ***reactive*** *if it has at least one possible reaction for each input in each state:* $\forall\, \mathbf{q}, i\,.\ \mid \delta_f(\mathbf{q}, i) \mid \geq 1$

Definition 4. ***Versatile machine***
A synchronous machine f is ***versatile*** *if it can produce all the possible outputs in each state:* $\forall\, \mathbf{q}, o\,.\, \exists\, i, \mathbf{q}'\,.\ \mathbf{q} \xrightarrow[o]{i} \mathbf{q}'$

Operations on synchronous machines

Definition 5. ***Projection***
Let f be a synchronous machine and $B' \subseteq B$ a subset of its output signals. The ***projected*** *machine $f_{\downarrow B'}$ is $(A, B', \mathbf{F}, \mathbf{i}_f, \Delta_{f_{\downarrow B'}})$ where*

$$\Delta_{f_{\downarrow B'}} = \left\{\ (\mathbf{q}, i, o \cap B', \mathbf{q}') \ \middle|\ \textit{for all}\ \ (\mathbf{q}, i, o, \mathbf{q}') \in \Delta_f\ \right\}$$

Definition 6. ***Synchronous product***
Let f (resp. g) be a synchronous machine with input signals in A (resp. C) and output signals in B (resp. D).
We define $(f \| g) = \big((A \backslash D) \cup (C \backslash B), B \cup D, \mathbf{F} \times \mathbf{G}, \mathbf{i}_f \mathbf{i}_g, \Delta_{(f \| g)}\big)$ where $\Delta_{(f \| g)}$ is defined by the inference scheme:

$$\frac{\mathbf{q}_1 \xrightarrow[b]{a \cup (d \cap A)} \mathbf{q}_1' \qquad \mathbf{q}_2 \xrightarrow[d]{c \cup (b \cap C)} \mathbf{q}_2'}{\mathbf{q}_1 \mathbf{q}_2 \xrightarrow[b \cup d]{a \cup c} \mathbf{q}_1' \mathbf{q}_2'}$$

where $a \subseteq A \backslash D$ and $c \subseteq C \backslash B$, $b \subseteq B$ and $d \subseteq D$

The synchronous product of synchronous machines f and g thus runs f and g in parallel, and figuratively plugs the outputs of f (resp. g) into the inputs of f (resp. g). One reaction step of the compound machine performs exactly one reaction step of f and one reaction step of g.

3 Traced categories

Traced monoidal categories —or rather traced categories, as the notion of trace relies on a monoidal structure— were introduced in [20] to give a categorical account of a situation arising in various areas of mathematics such as knot theory, algebraic topology or theoretical computer science.

3.1 Symmetric monoidal categories

Definition 7. ***Monoidal category***
A ***monoidal category*** *is a category* $\mathbf{C}$ *together with a bifunctor* $\otimes$*, a distinguished unit object* $\mathbf{1}$ *and natural families of isomorphisms*

- $\rho_A : A \otimes \mathbf{1} \xrightarrow{\sim} A$
- $\lambda_A : \mathbf{1} \otimes A \xrightarrow{\sim} A$
- $\alpha_{A,B,C} : A \otimes (B \otimes C) \xrightarrow{\sim} (A \otimes B) \otimes C$

satisfying some coherence axioms [11].
If these isomorphisms turn out to be identities —i.e. $A \otimes (B \otimes C) = (A \otimes B) \otimes C$ *and* $A \otimes \mathbf{1} = A = \mathbf{1} \otimes A$ *— the category is said to be* ***strict monoidal****.*

The strict case allows to write simpler, more readable equations. It turns out that every monoidal category is equivalent with a strict one [11], so we will most of the time avoid writing brackets around $\otimes$ without any loss of rigor.

Definition 8. ***Symmetries***
A family of ***symmetries*** *in a (strict) monoidal category* $\mathbf{C}$ *is a natural family of isomorphisms* $\sigma_{A,B} : A \otimes B \xrightarrow{\sim} B \otimes A$ *such that*

- $\sigma_{A,B} \,;\, \sigma_{B,A} = Id_{A \otimes B}$
- $\sigma_{A \otimes B,C} = (Id_A \otimes \sigma_{B,C}) \,;\, (\sigma_{A,C} \otimes Id_B)$

We say that $\mathbf{C}$ *is a (strict)* ***symmetric monoidal category*** *(****SMC****, for short) if* $\mathbf{C}$ *is a (strict) monoidal category together with a family of symmetries.*

3.2 Graphical language and the notion of trace

Along with the notion of traced category, [20] introduced a graphical language for these categories. As we shall see, the expression of some simple properties can be quite hard to read in the traditional equational language, while they look obvious in the graphical language. Figure 2 gives the translation of these properties to diagrammatical language.

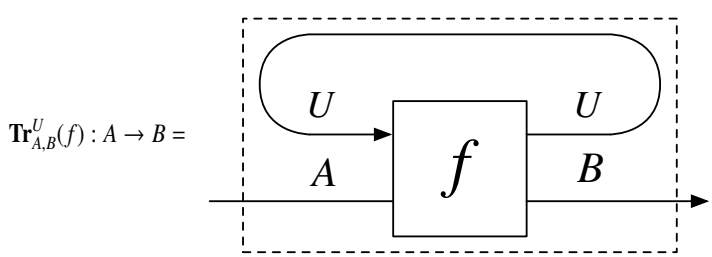


Fig. 1. Feedback: the trace

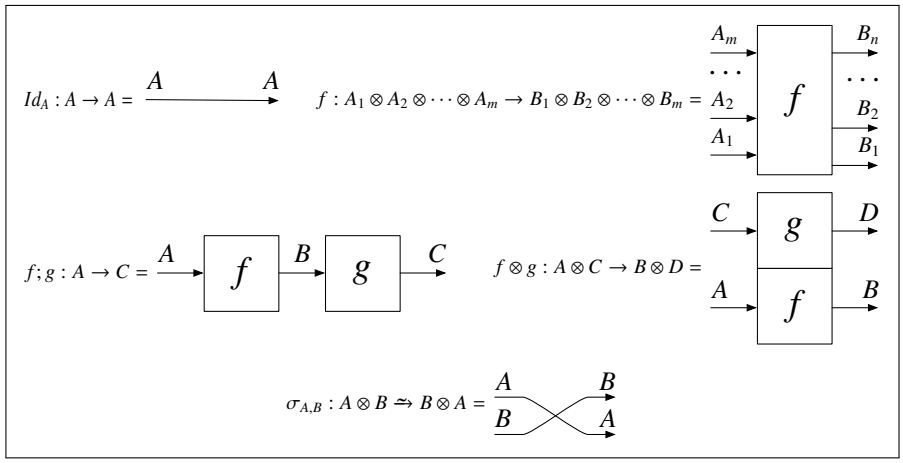


Fig. 2. The graphical language of symmetric monoidal categories

Once introduced this language of "circuits and wires", it is easy to state the basic idea behind the notion of trace: one wants to make (categorical) sense of the wiring presented on figure 3.2, which makes sense from a "circuit" perspective: just add a feedback wire and observe what happens[1].

The point is therefore to axiomatize the notion of *feedback* in a SMC.

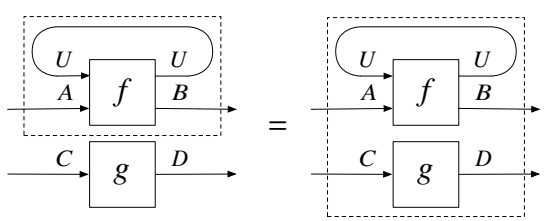
Definition 9. ***Trace***
A ***trace*** *in a (strict) SMC* $\mathbf{C}$ *is a family of functions*
(we will often omit the " $_{A,B}$ *" subscript when it is obvious from the context)*

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

satisfying the following axioms:

- ***Superposing:*** *for all* $f \ : \ A \otimes U \to B \otimes U$ *and* $g \ : \ C \to D$ *,*

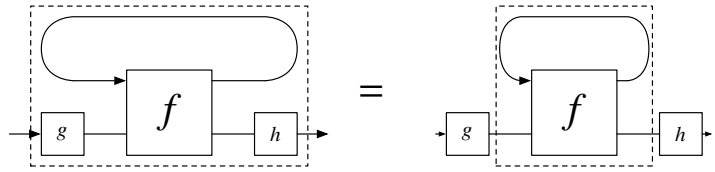
$$\mathbf{Tr}^U_{A,B}(f) \otimes g = \mathbf{Tr}^U_{A \otimes C, B \otimes D}(f \otimes g)$$



- ***Tightening:*** *for all* $f \ : \ A \otimes U \to B \otimes U$ *,* $g \ : \ A' \to A$ *and* $h \ : \ B \to B'$ *,*

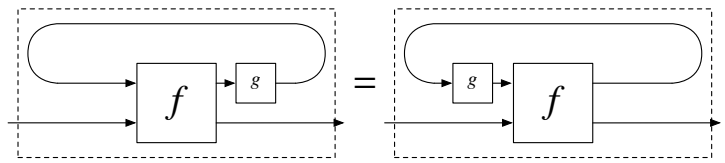
$$\mathbf{Tr}^U_{A',B'}\big((Id_U \otimes g) \,;\, f \,;\, (Id_U \otimes h)\big) = g \,;\, \mathbf{Tr}^U_{A,B}(f) \,;\, h$$

[1] This will eventually cause a fire, but this is off the scope of this work...



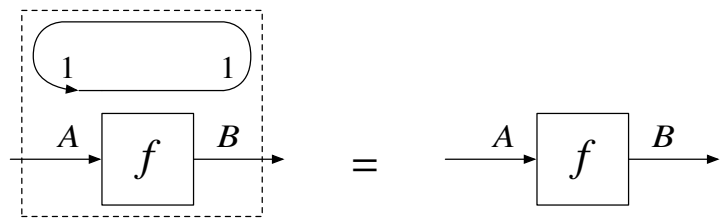
- ***Sliding:*** *for all* $f : A \otimes U' \to B \otimes U$ *and* $g : U \to U'$,

$$\mathbf{Tr}^{U}_{A,B}\big(f\,;(g \otimes Id_B)\big) = \mathbf{Tr}^{U'}_{A,B}\big((g \otimes Id_A)\,;f\big)$$



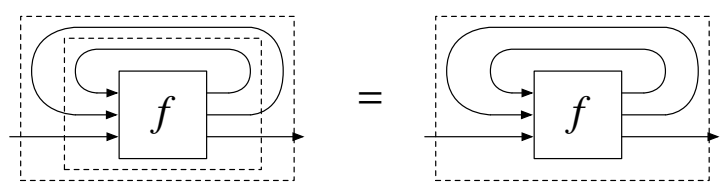
- ***Vanishing:*** *for all* $f : A \to B$ *(as the category is strict we have* $A = A \otimes \mathbf{1}$ *and* $B = B \otimes \mathbf{1}$ *)*

$$f = \mathbf{Tr}^{\mathbf{1}}_{A,B}(f)$$



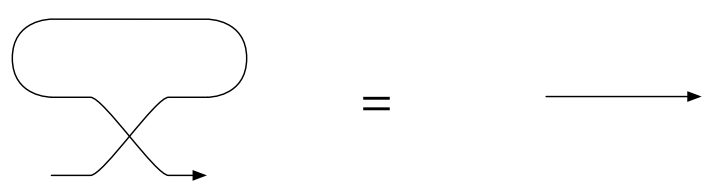
- ***Associativity:*** *for all* $f : A \otimes U \to B \otimes U$,

$$\mathbf{Tr}^{U}_{A,B}\left(\mathbf{Tr}^{V}_{A\otimes U, B\otimes U}(f)\right) = \mathbf{Tr}^{U\otimes V}_{A,B}(f)$$



- ***Yanking:*** *for all* A ,

$$\mathbf{Tr}^{A}_{A,A}(\sigma_{A,A}) = Id_A$$



We will refer to SMC equipped with a trace as ***traced categories****.*

3.3 The $\mathcal{G}$ construction

The **Int** construction was defined in [20] to relate traced categories with compact-closed categories. In [17], S. Abramsky introduced a (isomorphic) variant of it, called the $\mathcal{G}$ construction to give a categorical version of J.-Y. Girard's Geometry of Interaction (see section 6).

Definition 10. ***Compact-closed category***
A ***compact-closed category*** *is a SMC in which for every object A there is dual object A^* and a pair of arrows $\eta_A : \mathbf{1} \to A^* \otimes A$ (called the* ***unit****) and $\varepsilon_A : A \otimes A^* \to \mathbf{1}$ (called the* ***co-unit****) satisfying*
(again we assume the category is strict for readability)

- $A = A \otimes \mathbf{1} \xrightarrow{Id_A \otimes \eta_A} A \otimes A^* \otimes A \xrightarrow{\varepsilon_A \otimes Id_A} \mathbf{1} \otimes A = A$ *is equal to* Id_A
- $A^* = \mathbf{1} \otimes A^* \xrightarrow{\eta_{A^*} \otimes Id_{A^*}} A^* \otimes A \otimes A^* \xrightarrow{Id_{A^*} \otimes \varepsilon_{A^*}} \mathbf{1} \otimes A^* = A^*$ *is equal to* Id_{A^*}

The idea of the **Int** construction is to build a compact-closed category out of a traced category $\mathbf{C}$, the composition in $\mathbf{Int}(\mathbf{C})$ being defined using the trace of $\mathbf{C}$. The same is true of $\mathcal{G}$.

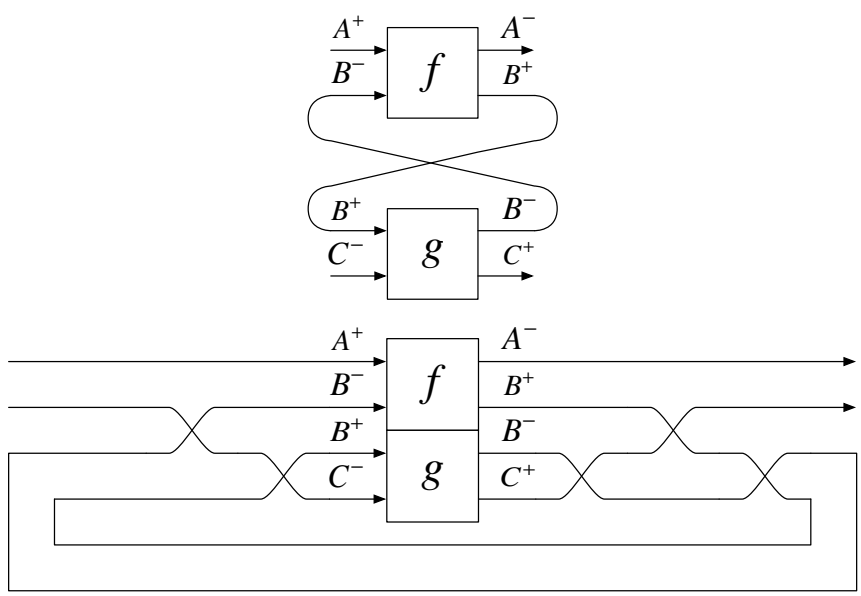


Fig. 3. Composition in $\mathcal{G}$, intuitive and formal diagrams

Definition 11. $\mathcal{G}$
If $\mathbf{C}$ *is a traced category, the category* $\mathcal{G}(\mathbf{C})$ *is defined as follows:*

- ***Objects*** *are pairs* A^+, A^- *of objects of* $\mathbf{C}$
- ***Morphisms*** $f : A^+, A^- \to B^+, B^-$ *in* $\mathcal{G}(\mathbf{C})$ *are given by morphisms* $f : A^+ \otimes B^- \to A^- \otimes B^+$ *in* $\mathbf{C}$
- ***Composition*** *of* $f : A^+, A^- \to B^+, B^-$ *and* $g : B^+, B^- \to C^+, C^-$ *is defined as*
  (see fig.3 for a more readable version in the graphical language)

$$f \,;_{\mathcal{G}}\, g := Tr^{B^+ \otimes B^-}_{A^+ \otimes C^-, A^- \otimes C^+} \left(S_1 \,;(f \otimes g)\,; S_2 \,;(Id_{A^-} \otimes Id_{C^+} \otimes \sigma_{(B^+, B^-)}) \right)$$

$$\text{where } S_1 := (Id_{A^+} \otimes \sigma_{C^-, (B^- \otimes B^+)}) \text{ and } S_2 := (Id_{A^-} \otimes \sigma_{(B^+ \otimes B^-), C^+})$$

- ***Identities*** Id_{A^+, A^-} *in* $\mathcal{G}(\mathbf{C})$ *are given by the symmetries* σ_{A^+, A^-} *of* $\mathbf{C}$

The fact that $\mathcal{G}(\mathbf{C})$ is indeed a category follows from the axioms of trace.

As the compact-closed structure of $\mathcal{G}(\mathbf{C})$ is not our main interest in this paper, we skip its definition. The interested reader may find it for instance in [8].

3.4 Diagonal morphisms

The tensor product of a SMC is not in general a cartesian product, which means basically that there is no *diagonal morphism* $c_A : A \to A \otimes A$ satisfying the right set of equation to talk about duplication of data.

However, one may want to identify among the morphisms of a given SMC the ones that can be manipulated as if the tensor product was cartesian.

Definition 12. ***Diagonals***
A (strict) SMC $\mathbf{C}$ *is said to have* ***diagonals*** *there exists two (not necessarily natural) families of morphisms* $c_A : A \to A \otimes A$ *and* $w_A : A \to \mathbf{1}$ *such that*

- (A, c_A, w_A) *is a symmetric comonoid for all* A*, that is to say:*
  $c_A \,;(Id_A \otimes c_A) = c_A \,;(c_A \otimes Id_A)$ *and* $c_A \,;(w_A \otimes Id_A) = c_A \,;(Id_A \otimes w_A) = Id_A$
- $w_1 = Id_1$ *and* $w_A \otimes w_B = w_{A \otimes B}$
- $c_A \otimes c_B \,;(Id_A \otimes \sigma_{A,B} \otimes Id_B) = c_{A \otimes B}$

In such a category, the ***weak initial morphism*** *is defined for all objects* A *as* $i_A := \mathbf{Tr}^A_{\mathbf{1},A}(c_a)$.

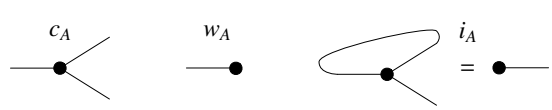
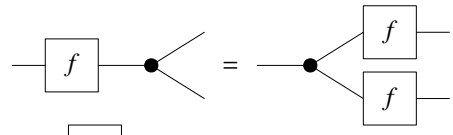
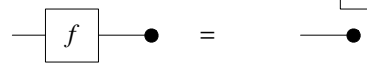
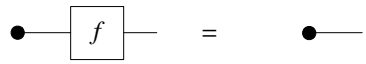


Fig. 4. The morphisms c_A, w_A, i_A in the graphical language

One can then define class of morphisms that behave well with respect to the diagonal structure.

Definition 13. ***Diagonal morphisms***
A morphism $f : A \to B$ *in a SMC with diagonals is said to be*

- ***copyable*** *if* $f\,;c_B = c_A\,;(f \otimes f)$ 
- ***discardable*** *if* $w_A = f\,;w_B$ 
- ***strict*** *if* $i_A\,;f = i_B$ 

Remark : *The morphisms of a SMC with diagonals that are both copyable and discardable form a subcategory, the largest one on which the tensor product is a cartesian product.*

4 The category of synchronous machines

We can see now how the synchronous machines can be organized as a traced category.

First definitions

The basic idea is that a synchronous reactive system is a morphism from its input set of signals to its output set of signals. In a way, the category we describe is quite close to the category **Rel** of sets and relations. The difference being the *state space*, which needs a specific treatment in terms of composition and equality.

Definition 14. ***Objects, morphisms***
The category **S** *of synchronous machines is defined as*

- ***objects*** *are sets*
- ***morphisms*** *from* A *to* B *are synchronous machines (definition 1) with input* A *and output* B

Given two morphisms in **S**, one has to be able to say when they are *equal*. In fact, with plain equality we would not end up with a category. The issue is that the state space has to be treated somehow *up to isomorphism*. This amounts to the standard issue of equality of labelled transition system. The notion we settle for in this paper is isomorphism of state graphs.

Definition 15. ***Equality***
Two morphisms $f, f' : A \to B$ *in* **S** *are equal whenever they are graph-isomorphic, i.e.*

$$\textit{there exists a bijection } \varphi : \mathbf{F} \xrightarrow{\sim} \mathbf{G} \textit{ such that } \Delta_{f'} = \varphi \circ \Delta_f \circ \varphi^{-1}$$

We can now define the composition of two morphisms f, g in $\mathbf{S}$. Look back at the graphical representation of composition in a monoidal category: the intuition is that when composing, the output of f is passed on as an input for g. As for the state space, it is natural to say that a state of the composed system is given by a state of f and a state of g, hence the use of cartesian product.

Definition 16. ***Composition and identities***
The composition of two morphisms $f : A \to B$ and $g : B \to C$ in $\mathbf{S}$ is given by

$$f \,;\, g \;:=\; \big(A \,,\, C \,,\, \mathbf{F} \times \mathbf{G} \,,\, \mathbf{i}_f \mathbf{i}_g \,,\, \Delta_{g\,;\,f} \big)$$

$$\textit{where} \quad \Delta_{f\,;\,g} \;:=\; \Big\{ \mathbf{fg} \xrightarrow[c]{a} \mathbf{f}'\mathbf{g}' \;\Big|\; \exists b \subseteq B \,,\, \mathbf{f} \xrightarrow[b]{a} \mathbf{f}' \;\; \textit{and} \;\; \mathbf{g} \xrightarrow[c]{b} \mathbf{g}' \Big\}$$

For any object A, the identity Id_A is defined as

$$Id_A \;:=\; \Big(A \,,\, A \,,\, \{*\} \,,\, * \,,\, \big\{ * \xrightarrow[a]{a} * \;\big|\; \textit{for all } a \subseteq A \big\} \Big)$$

It is routine to check that all this defines a category: first check that equality is compatible with composition, then proving associativity and identity axioms is just a matter of choosing appropriate (and trivial in that case) bijections between state spaces.

Symmetric monoidal structure

The tensor product is the operation that puts two (non-interacting) systems in parallel. Again the graphical language suggests the use of the cartesian product for handling the state space.

Definition 17. ***Tensor product***
On objects of $\mathbf{S}$, the tensor product is the disjoint union, $A \otimes B := A \uplus B$.
If $f : A \to B$ and $g : C \to D$ are morphisms in $\mathbf{S}$, their tensor product is given by

$$f \otimes g \;:=\; \big(A \uplus B \,,\, C \uplus D \,,\, \mathbf{F} \times \mathbf{G} \,,\, \mathbf{i}_f \mathbf{i}_g \,,\, \Delta_{f \otimes g} \big)$$

$$\textit{where} \quad \Delta_{f \otimes g} \;:=\; \Big\{ \mathbf{fg} \xrightarrow[bd]{ac} \mathbf{f}'\mathbf{g}' \;\Big|\; \mathbf{f} \xrightarrow[b]{a} \mathbf{f}' \;\; \textit{and} \;\; \mathbf{g} \xrightarrow[d]{c} \mathbf{g}' \Big\}$$

The unit $\mathbf{1}$ of the tensor product is the empty set.

Definition 18. ***Symmetries***
Given two object A, B of $\mathbf{S}$, the associated symmetry $\sigma_{A,B} : A \otimes B \xrightarrow{\sim} B \otimes A$ is defined as

$$\sigma_{A,B} \;:=\; \Big(A \uplus B \,,\, B \uplus A \,,\, \{*\} \,,\, * \,,\, \big\{ * \xrightarrow[ba]{ab} * \;\big|\; \textit{for all } a \subseteq A \,,\, b \subseteq B \big\} \Big)$$

The trace

The trace is quite similar to the one of $\mathbf{Rel}$. Note that it does neither preserve reactivity nor determinism of synchronous machines.

Definition 19. ***Trace***
Given a morphism $f : A \otimes U \to B \otimes U$ of $\mathbf{S}$, its trace $\mathbf{Tr}^U_{A,B}(f) : A \to B$ is defined as

$$\mathbf{Tr}^U_{A,B}(f) := \left(A\,,\, B\,,\, \mathbf{F}\,,\, \mathbf{i}_f\,,\, \left\{ \mathbf{f} \xrightarrow[b]{a} \mathbf{f}' \;\middle|\; \exists u \subseteq U\,,\, \mathbf{f} \xrightarrow[bu]{au} \mathbf{f}' \right\} \right)$$

Copyable, discardable and strict morphisms

Proposition 1. ***Diagonal morphisms in*** $\mathbf{S}$
The category $\mathbf{S}$ has diagonals, given by

$$c_A := \left(A\,,\, A \uplus A\,,\, \{*\}\,,\, *\,,\, \left\{ * \xrightarrow[aa]{a} * \mid \textit{for all } a \subseteq A \right\} \right)$$

$$w_A := \left(A\,,\, \mathbf{1}\,,\, \{*\}\,,\, *\,,\, \left\{ * \xrightarrow[\emptyset]{a} * \mid \textit{for all } a \subseteq A \right\} \right)$$

Moreover, the weak initial morphism for an object A is

$$i_A = \mathbf{Tr}^A_{\mathbf{1},A}(c_a) = \left(\mathbf{1}\,,\, A\,,\, \{*\}\,,\, *\,,\, \left\{ * \xrightarrow[a]{\emptyset} * \mid \textit{for all } a \subseteq A \right\} \right)$$

Remark : *weak initial morphisms are quite dramatic examples of the non-preservation of determinism by the trace of $\mathbf{S}$.*

Theorem 1. ***Deterministic, total and strict morphisms in*** $\mathbf{S}$

- *the copyable morphisms are the* deterministic, one-state *synchronous machines*
- *the discardable morphisms are the* reactive, one-state *synchronous machines*
- *the total morphisms are the* surjective, one-state *synchronous machines*

5 $\mathcal{G}(\mathbf{S})$ and the synchronous product

We now relate the composition in $\mathcal{G}(\mathbf{S})$ with the synchronous product of synchronous machines.
The theorem below says that, provided the input/output sets of the two systems are "correctly positioned"[2], their composition in $\mathcal{G}(\mathbf{S})$ is equal to their synchronous product followed by hiding the signal used during the interaction.

[2] That is, they do not intersect anywhere but on B^+, B^- where the systems are expected to interact.

Theorem 2. *Let* $f : (A^+, A^-) \to (B^+, B^-)$ *and* $g : (B^+, B^-) \to (C^+, C^-)$ *be two morphisms is* $\mathcal{G}(\mathbf{S})$. *Suppose also that* $A^+ \cap C^+ = A^- \cap C^- = \emptyset$ (see discussion below). *Then we have*

$$f \,;_{\mathcal{G}}\, g \;=\; (f \| g)_{\downarrow A^- \uplus C^+}$$

Proof. $f \,;_{\mathcal{G}}\, g$ and $(f \| g)_{\downarrow A^-, C^+}$ share the same state space $\mathbf{F} \times \mathbf{G}$ and initial state $\mathbf{i}_f \mathbf{i}_g$. Therefore we show they are equal (definition 15) by the *identity* bijection between their state spaces:

We have that $\mathbf{fg} \xrightarrow[a'c']{ac} \mathbf{f'g'}$ in $f \,;_{\mathcal{G}}\, g$

if and only if there exist b^+, b^- such that $\mathbf{fg} \xrightarrow[a'c'b^+b^-]{acb^+b^-} \mathbf{f'g'}$ in $S_1 \,;\, (f \otimes g) \,;\, S_2 \,;\, (\mathrm{Id}_{A^-} \otimes \mathrm{Id}_{C^+} \otimes \sigma_{(B^+,B^-)})$,
(with $S_1 := (Id_{A^+} \otimes \sigma_{(B^+ \otimes B^-),C^-})$ *and* $S_2 := (Id_{A^-} \otimes \sigma_{(B^+ \otimes B^-),C^+})$ *as in definition 11)*

if and only if there exist b^+, b^- such that $\mathbf{fg} \xrightarrow[a'b^+b^-c']{ab^-b^+c} \mathbf{f'g'}$ in $f \otimes g$,

if and only if there exist b^+, b^- such that $\mathbf{f} \xrightarrow[a'b^-]{ab^+} \mathbf{f'}$ in f and $\mathbf{g} \xrightarrow[b^+c']{b^-c} \mathbf{g'}$ in g,
if and only if there exist b^+, b^- such that $\mathbf{fg} \xrightarrow[ab^+b^-c']{ac} \mathbf{f'g'}$ in $(f \| g)$,

if and only if $\mathbf{fg} \xrightarrow[a'c']{ac} \mathbf{f'g'}$ in $(f \| g)_{\downarrow A^- \uplus C^+}$. □

The condition $A^+ \cap C^+ = A^- \cap C^- = \emptyset$ may look simply technical. But there is actually more to it. As explained in section 2, synchronous programming languages such as Esterel often assume that there is a global space of signals $\mathbb{S}$. Reactive systems do not have a specified input/output space and they rather all read and emit signals in $\mathbb{S}$. If two systems are executed side by side and share input/output signal, they will interfere.

This does not quite correspond to the categorical point of view, which works *up to isomorphism*: to get a monoidal category, one needs to avoid *interference* between f and g when building $f \otimes g$. This is achieved using direct sums on input/output spaces of reactive systems when defining $\otimes$, while with a global signal set $\mathbb{S}$ one would use union, with possible interference so that the $\otimes$ would be partially defined.

The same dilemma occurs in logics, this is the theme of **locativity**, introduced by J.-Y. Girard in [6]. Locative approaches to logics start by setting a global space where everything happens, and also use union rather than direct sum, ending up with partially defined connectives.

From this point of view, we could say that synchronous machines are naturally *locative*.

6 Related and future work

Circuit synthesis

The *geometry of synthesis* research program by Ghica et al. [2] proposes novel ways of compiling an idealized high-level language to digital circuits. The language a dialect of Algol, that is an higher-order functional languages with imperative features, extended with finite-state asynchronous concurrency. Various type systems have been proposed to accommodate this finiteness condition.
Part of the originality of the approach stems from the origins of the translation process, heavily inspired from categorical models. Compared with most works in high-level circuit synthesis, this denotational flavor gives a compositional translation, i.e. one where functions can be compiled separately.
Following the seminal work of Alur et al. [9], Ghica et al. also studied the *round abstraction* [12], a transformation from asynchronous to synchronous circuits, in order to benefit from the rich ecosystem available for the latter.
One possible application of the present work would be to be able to directly generate synchronous circuits from a high-level programming language. The

Towards geometry of interaction for synchronous machines

The $\mathcal{G}$ construction was first introduced to give a categorical account of the Geometry of Interaction program. This program that was initiated and pursued by J.-Y. Girard in a series of paper [4,5,7]. It is built around the idea of looking at the *interactive* aspects of logical phenomena.
The composition of two proofs —the `(Cut)` rule— is seen interactively in GoI: the computation can flow back and forth between the two composed programs, as in game semantics [13] for instance.
Moreover, in the GoI approach, for a program to be of a certain type is no longer a matter of following some given set of rules, but rather the result of mutual testing. In logics, this technique is usually referred to as construction by orthogonality.
By showing that synchronous machines form a traced category and that the synchronous product corresponds to the composition induced by the $\mathcal{G}$ construction, our work opens the way to applying GoI ideas and techniques to reactive systems. In particular, it would allow to use the good structural properties obtained by orthogonality methods to try to deal with the problem of determinism/reactiveness.

References

1. Gérard Berry ; Laurent Cosserat, *The esterel synchronous programming language and its mathematical semantics*, Seminar on Concurrency, Carnegie-Mellon University (London, UK, UK), Springer-Verlag, 1985, pp. 389–448.
2. Dan R. Ghica, *Function interface models for hardware compilation*, MEMOCODE, 2011, pp. 131–142.

3. Jean-Yves Girard, *Linear logic*, Theoretical Computer Science **50** (1987), no. 1, 1 – 101.
4. _____, *Towards a Geometry of interaction*, Contemporary Mathematics n° 92, American Mathematical Society, 1989, pp. 69 – 108.
5. _____, *Geometry of interaction II : deadlock-free algorithms*, Lecture Notes in Computer Science n° 417 (Martin-Löf and Mints, eds.), Springer-Verlag, 1990, pp. 76 – 93.
6. _____, *Locus solum: From the rules of logic to the logic of rules*, Mathematical Structures in Computer Science **11** (2001), no. 3, 301–506.
7. _____, *Geometry of interaction V: Logic in the hyperfinite factor*, Theor. Comput. Sci. **412** (2011), no. 20, 1860–1883.
8. Esfandiar Haghverdi, *A categorical approach to linear logic, geometry of proofs and full completeness*, Ph.D. thesis, University of Otawa, 2000.
9. Rajeev Alur ; Thomas A. Henzinger, *Reactive modules*, Formal Methods in System Design **15** (1999), no. 1, 7–48.
10. Albert Benveniste ; Paul Le Guernic ; Christian Jacquemot, *Synchronous programming with events and relations: the SIGNAL language and its semantics*, Science of Computer Programming **16** (1991), 103–149.
11. Saunders Mac Lane, *Categories for the working mathematician*, Graduate Texts in Mathematics, no. 5, Springer-Verlag, 1971.
12. Dan R. Ghica ; Mohamed N. Menaa, *On the compositionality of round abstraction*, CONCUR, 2010, pp. 417–431.
13. Samson Abramsky ; Guy McCusker, *Game semantics*, Logic and Computation : Proceedings of the 1997 Marktoberdorf Summer School, Springer-Verlag, 1998.
14. Paul Caspi ; Daniel Pilaud ; Nicolas Halbwachs ; John A. Plaice, *Lustre: a declarative language for real-time programming*, POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (New York, NY, USA), ACM, 1987, pp. 178–188.
15. David Harel ; Amir Pnueli, *On the development of reactive systems*, Logics and models of concurrent systems (Krzysztof R. Apt, ed.), Springer-Verlag New York, Inc., New York, NY, USA, 1985, pp. 477–498.
16. Nicolas Halbwachs ; Fabienne Lagnier ; Pascal Raymond, *Synchronous observers and the verification of reactive systems*, Proceedings of the Third International Conference on Methodology and Software Technology: Algebraic Methodology and Software Technology (London, UK, UK), AMAST '93, Springer-Verlag, 1994, pp. 83–96.
17. Samson Abramsky ; Esfandiar Haghverdi ; Philip Scott, *Geometry of interaction and linear combinatory algebras*, Mathematical. Structures in Comp. Sci. **12** (2002), no. 5, 625–665.
18. Albert Benveniste ; Paul Caspi ; Stephen A. Edwards ; Nicolas Halbwachs ; Paul Le Guernic ; Robert De Simone, *The synchronous languages twelve years later*, Proceedings of the IEEE, 2003, pp. 64–83.
19. Jean-Yves Girard ; Yves Lafont ; Paul Taylor, *« Proofs and Types »*, Cambridge University Press, 1989.
20. André Joyal ; Ross Street ; Dominic Verity, *Traced monoidal categories*, Mathematical Proceedings of the Cambridge Philosophical Society, 119, pp. 447-468, 1996.