The Geometry of Synchronization

Ugo Dal Lago, Claudia Faggian, Ichiro Hasuo, Akira Yoshimizu

The Geometry of Synchronization

Ugo Dal Lago
University of Bologna & INRIA
ugo.dallago@unibo.it

Claudia Faggian
CNRS
faggian@pps.univ-paris-diderot.fr

Ichiro Hasuo
Akira Yoshimizu
University of Tokyo
{ichiro, yoshimizu}@is.s.u-tokyo.ac.jp

Abstract

We graft synchronization onto Girard's Geometry of Interaction in its most concrete form, namely token machines. This is realized by introducing proof-nets for SMLL, an extension of multiplicative linear logic with a specific construct modeling synchronization points, and of a multi-token abstract machine model for it. Interestingly, the correctness criterion ensures the absence of deadlocks along reduction and in the underlying machine, this way linking logical and operational properties.

Categories and Subject Descriptors F.3.2 [*Logics and Meaning of Programs*]: Semantics of Programming Languages — Operational Semantics.; F.4.1 [*Mathematical Logic and Formal Languages*]: Mathematical Logic — Proof Theory.

General Terms Theory

Keywords geometry of interaction, token machines, proof-nets, linear logic, quantum computation

1. Introduction

One of the reasons making linear logic [11] a breakthrough not only in proof theory but also in programming language semantics, is that it enables an interactive view of computation through Game Semantics and the Geometry of Interaction (GoI in the following). This way, proofs and higher-order programs are seen as mathematical objects with a rich interactive behaviour (composition in the former, execution in the latter). One could say that Game Semantics focuses on the interpretation of programs, the tool of election being full abstraction, while Geometry of Interaction is a fine-grained model of computation itself, and provides insights on quantitative aspects of computation [2, 4], tools to allow optimization [15] and guidelines in the design of a compiler [20, 22]. In either settings, one can describe the dynamics of the interaction between programs and their environments in several ways. In Game Semantics, this can take a categorical form, or the operational form of an abstract machine, as in [3]. Similarly in GoI, the interaction can be described either via automata [9], or via traced monoidal categories [1]. GoI can also be presented algebraically, by way of operator algebras [14], or more operationally as an algebra of clauses [12]. Different presentations suit different aims.

CSL-LICS 2014, July 14–18, 2014, Vienna, Austria.

ACM 978-1-4503-2886-9...$15.00.
http://dx.doi.org/10.1145/2603088.2603154

In this paper, we are interested in the most concrete presentation of GoI, and in particular in the so-called Interaction Abstract Machines (IAMs in the following). IAMs are bi-deterministic automata by which one interprets λ-terms in such a way that β-equivalent terms are interpreted by equivalent IAMs (*i.e.*, IAMs computing the same function). A single run of the IAM interpreting a λ-term does *not* suffice to capture completely the behaviour of the term itself: this in general requires multiple calls to the IAM, which however can proceed in parallel, without any need for synchronization. In a sense, this shows that GoI has the potential to somehow capture the inherent parallelism of functional programs, and this has been indeed exploited as a compilation technique through (directed) virtual reduction [21, 22] (even though such work deviates from purely interactive machines). The captured kind of parallelism is however lacking a fundamental ingredient, since the parallel components are not allowed to interact in non-trivial ways.

This work is a study of synchronization in the context of linear logic proofs. More specifically, the contributions of this paper are threefold:

- Proof-nets of multiplicative linear logic (MLL in the following) are enriched so as to include a specific rule representing synchronization points. This is done by extending the kinds of links on top of which proof-structures are defined, then properly adapting Danos and Regnier's correctness criterion. The resulting system, called SMLL, is shown to enjoy cut-elimination.
- A specific kind of Interaction Abstract Machine, called SIAM, is introduced and shown to be a model of SMLL. Remarkably, SMLL nets have the property that the underlying SIAM is deadlock-free.
- SMLL is shown to be sufficiently rich to interpret a quantum λ-calculus akin to those recently introduced by Selinger and Valiron [24]. Synchronization plays the essential role of reflecting quantum entanglement, itself a crucial ingredient for the efficiency of quantum computation [17]. This requires to extend SMLL only slightly, by endowing proof-structures with quantum registers, but keeping the underlying logical structure essentially unchanged.

An extended version of this paper with more details is available [5].

2. Linear Logic and Token Machines

In this section, we will give some hints about how IAMs (close variations of which include token machines [20] and context semantics [9]) are defined, pointing to the relevant literature on the subject.

Let's start with a linear, simply-typed, λ-calculus. Even in the absence of constants, the language has a decent expressive power [25]: all boolean circuits can be encoded into it. Booleans can be encoded as the two permutations on a two-element set: `true` is the λ-term $\lambda\langle x, y\rangle.\langle x, y\rangle$, while `false` is the λ-term

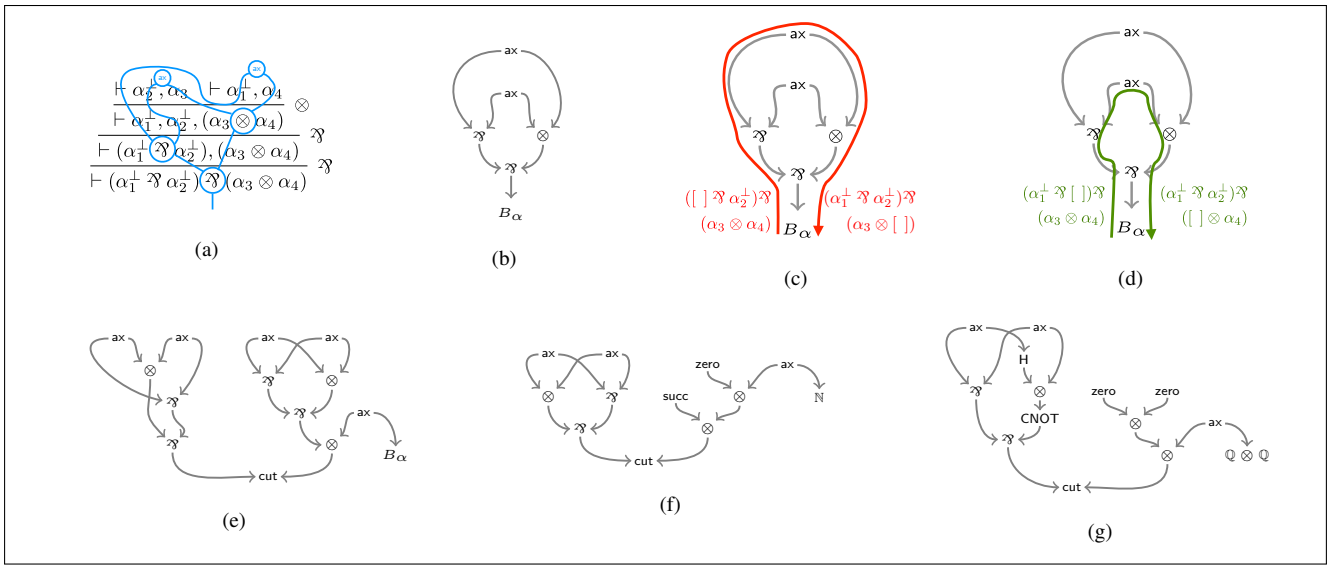


Figure 1. Nets and Abstract Machines — Some Examples.

$\lambda\langle x,y\rangle.\langle y,x\rangle$; both can be given the same type $B_\alpha = \alpha \otimes \alpha \multimap \alpha \otimes \alpha = (\alpha^\perp \parr \alpha^\perp) \parr (\alpha \otimes \alpha)$. Boolean functions can also be represented in the calculus. As a simple example, consider the combinator $\texttt{not} = \lambda x.\lambda\langle y,z\rangle.x\langle z,y\rangle$. As can be easily verified, the application $\texttt{not}\ \texttt{true}$ has type B_α and β-reduces to $\texttt{false}$. There's a different, "reduction-free" way to compute the result of the application $\texttt{not}\ \texttt{true}$: traveling inside (a graph-based representation of) the term. Indeed, the type derivation for $\texttt{false}$ in Figure 1(a) can be represented as the proof-net in Figure 1(b). If we start from the *leftmost* negative occurrence of α in its conclusion and track it in the natural way, we end up in the *rightmost* positive occurrence of α in the conclusion (Figure 1(c)). Similarly, if we start from the rightmost negative occurrence of α, we arrive at the leftmost positive occurrence of α (Figure 1(d)). As expected, the term $\texttt{true}$ behaves the opposite. Generalizing a bit, this game reveals the shape of normal forms, and the nice thing is that it can be played on terms which *are not* in normal form, this way becoming a fully-fledged notion of computation. As an example, consider the proof-net in Figure 1(e) which corresponds to the term $\texttt{not}\ \texttt{true}$, and the paths (like in Figure 1(c)) certifying that the term rewrites to $\texttt{false}$. The way one traces atom occurrences along paths can be formalized as an automaton, the Interaction Abstract Machine (IAM in the following), whose states are atom occurrences which are associated with the edges of the graph, and whose transitions only depend on the nodes in the underlying graph (the transitions for MLL can be found in the first two rows of Figure 6).

The ideas above have been extensively developed. In particular, Interaction Abstract Machines have been defined for λ-calculi in which duplication is indeed possible, and also for *applicative* λ-calculi, *i.e.*, calculi endowed with constants and possibly recursion. Take, as an example, the simple PCF term $t = (\lambda x.\lambda y.xy)\ S\ \underline{0}$ where $S\colon \mathbb{N} \multimap \mathbb{N}$ and $\underline{0}\colon \mathbb{N}$ are constants for successor and zero, respectively. The fact that the term above evaluates to 1, again, can be observed by letting *one* token travel inside the (proof-net corresponding to a) type derivation for t, as depicted in Figure 1(f). The token now starts its journey from the node labeled with zero carrying the natural number "built so far", which initially is of course 0. After some re-routing induced by the multiplicative nodes $\parr$ and $\otimes$, the token reaches the node labelled with succ, which modifies the natural number to 1, and the journey proceeds until the conclusion.

How about quantum computation? Would it be possible to adapt the scheme above to λ-calculi specifically designed for quantum computation? Token machines seem to be a natural way to model inherently linear calculi such as quantum λ-calculi. However, if one tries to *directly* apply the paradigm described above, one soon gets into troubles. Consider, as an example, a term like $u = (\lambda\langle x,y\rangle.\mathsf{CNOT}\langle \mathsf{H}x, y\rangle)\langle \underline{0}, \underline{0}\rangle$, where CNOT and H are certain unitary operations that act on 2-qubit and 1-qubit systems, respectively. This is an encoding of a quantum circuit having the remarkable property of producing an entangled pair of qubits in output. Let us try to play the same game we played with t on a graph-theoretic representation of u (see Figure 1(g)). If we allow a token to start its journey from the leftmost occurrence of $\underline{0}$, it can of course reach H, go through it (having the underlying qubit modified accordingly) but gets stuck at CNOT. Indeed, the value of the first and second outputs of CNOT can only be known when *both* inputs are available. But even more importantly, the state of those qubits is an entangled state, *i.e.*, it cannot be described as the tensor product of the two qubits. Switching to a setting in which an IAM state consists not of a single token but possibly of multiple ones seems very natural now. Moreover, as is clear from the CNOT example, there should be a way to force those many tokens to *synchronize*, *i.e.*, to wait until some of the other tokens reaches a certain state, before proceeding.

In all the "non quantum" examples, *multiple* tokens could travel the net in parallel, but what we compute is exactly the same, because the tokens do not interact with each other. Things are very different in the quantum example. Summing up, as computational models the GoI machines are effective, powerful (as shown by the results on optimal reductions and implicit complexity from the literature). However, they have limits:

- a *general* limit in expressiveness, as they capture parallel computation, but without synchronization;
- a *specific* limit, as it is not possible to model quantum computing without some form of synchronization.

Synchronizations and Deadlocks. As the reader may expect, the most delicate property in a multi-token setting is deadlock freedom. Consider Figure 2. Squares connected with lines represent

synchronization points: tokens should cross simultaneously s_1 and cross simultaneously s_2. In this configuration, if we have a token on each of the three positions $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$, they are in deadlock. In the following, we develop an approach which reduces the absence of deadlocks in the machine M_R interpreting a proof-net R to the correctness criterion of R, our *motto* then being the following:

"Correctness of R" $\Rightarrow$ "Deadlock Freedom of M_R".

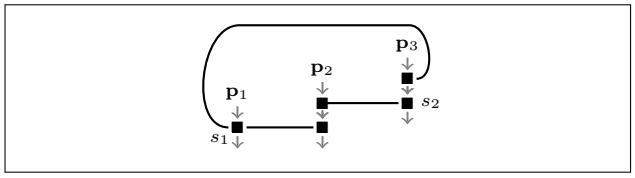


Figure 2. Deadlocked Structures.

Some Related Work. The initial motivation leading us to the development of Geometry of Synchronization was the study of quantum computation.

Multiple-token machines have already been investigated by the first author and Margherita Zorzi in a recent unpublished manuscript [6]. In this paper, we take a step back and analyze the construction from a logical point of view, showing how parallelism and synchronization can be satisfactorily captured within a slight variation of multiplicative linear logic.

A version of quantum proof-nets have recently been proposed by the authors [26]. Boxes are seen as a way to implement quantum measurement, and the reader can find several useful examples there. The proposed class of proof-nets has weaker properties, however: the results on cut-elimination are rather limited, and do not allow us to study deadlock freedom as we do here. Having better proof theoretical properties, as in the nets in this paper, allows us to go further in the interpretation of quantum λ-calculi, which we are able to simulate in a sound way. Moreover, we uncover and separate the classic computational structure, making it independent from quantum data. Not only this allows to modularize the results, but we believe that other interesting application examples for the SIAM, besides quantum computing, can be found, such as distributed implementations (see, *e.g.*, [10]).

We finally like to mention other work which is related to SMLL in that the aim is to capture synchronization into a logical calculus, in particular [16] and [7].

3. SMLL

In this section we introduce SMLL nets, which are a generalization of proof-nets for multiplicative linear logic. As a reference to proof-nets, we suggest [18] — our approach to correctness is close to the one described there.

Formulas. The language of SMLL *formulas* is identical to the one for MLL, *i.e.*,

$$A ::= 1 \;\Big|\; \bot \;\Big|\; X \;\Big|\; X^{\bot} \;\Big|\; A \otimes A \;\Big|\; A \parr A,$$

where X ranges over a denumerable set of *propositional variables*. The constants $1, \bot$ are the *units*. We call *atomic* those formulas which are either (possibly negated) propositional variables or units. Linear negation $(\cdot)^{\bot}$ is extended into an involution on all formulas as usual: $X \equiv X^{\bot\bot}$, $1^{\bot} \equiv \bot$, $\bot^{\bot} \equiv 1$, $(A \otimes B)^{\bot} \equiv A^{\bot} \parr B^{\bot}$, $(A \parr B)^{\bot} \equiv A^{\bot} \otimes B^{\bot}$. Linear implication is a defined connective: $A \multimap B \equiv A^{\bot} \parr B$.

Polarized Formulas. Atoms and connectives of MLL are divided in two classes: positive $(1, X, \otimes)$ and negative $(\bot, X^{\bot}, \parr)$. In this paper, to have a compact presentation, we exclude from polarized formulas the propositional variables, and define *positive* formulas (denoted by P) and *negative* formulas (denoted by N) as follows:

$$P ::= 1 \;\Big|\; P \otimes P; \qquad N ::= \bot \;\Big|\; N \parr N.$$

Observe that the formula $A = 1 \multimap 1 \equiv \bot \parr 1$ contains one negative occurrence of atom and one positive occurrence of atom; thus A, in our setting, is neither positive nor negative.

3.1 Structures

An SMLL *structure* is a labeled directed (multi-)graph, where the edges are labeled with MLL formulas (the label of an edge is called its *type*). The alphabet of *nodes*, given in Figure 3(a-c), is the same as the one of MLL, but extended with a new link, called a *sync link*. Altogether, we have MLL links (Figure 3(a)), unit links (Figure 3(c)), and sync links (Figure 3(b)), which we detail below.

Graphically, we represent structures with the edges oriented from top to bottom; we use accordingly terms like "above", "below", "upwards" and "downwards". We call *conclusions* (resp. *premisses*) of a link those edges represented below (resp. above) the link symbol. The sort of a link induces constraints on the number

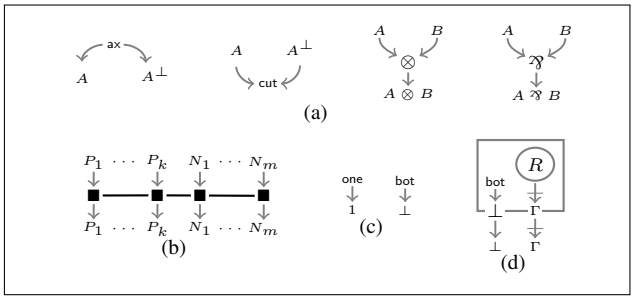


Figure 3. SMLL Links.

and the labels of its premisses and conclusions, as shown in Figure 3. The graph can have pending edges, *i.e.*, some edges may not have a target; the pending edges are called the *conclusions* of the structure. We will often say that a link "has a conclusion (premiss) A" as shortcut for "has a conclusion (premiss) of type A". When we need more precision, we distinguish between an edge and its type, and we use variables such as e, f for the edges.

Sync links. A sync link has n premisses, and n conclusions. For each i $(1 \leq i \leq n)$ the i-th premiss e_i and the *corresponding* i-th conclusion f_i are typed by the *same* formula, which is either positive or negative. To stress the correspondence between the i-th premiss and the i-th conclusion, we find it convenient to graphically represent an n-ary sync link as a list of n nodes connected by untyped edges. In the example in Figure 3(b) , we have $n = k + m$ edges, which are typed with k positive and m negative formulas.

We now need some specific terminology. An edge is *positive* (*negative*) if its type is positive (or negative); we also say that such an edge is *polarized*. A node is polarized if its conclusions are all polarized. All edges of sync links are polarized; we will borrow some of the terminology from polarized linear logic [19]. Given a sync link, we call *in-edges* its positive premisses and negative conclusions, and call *out-edges* the positive conclusion and negative premisses.

Intuitively, a sync link acts on an edge of type A, but does not introduce A. We call *sync path* a path which traverses only sync links, going in and coming out on corresponding edges. We say that an edge e of type A is a *hereditary conclusion* of a link l if there is a sync path from the conclusion of l to e .

Units and the Unit-Free Fragment. The units are 1 and $\bot$, respectively introduced by the links one and bot. To the bot link is associated a notion of box (Figure 3(d)), which we discuss next.

We indicate by SMLL^0 the fragment of SMLL without unit links (therefore in particular without boxes); in SMLL^0, the formulas 1 and $\bot$ are hence only introduced as conclusions of axioms. Even if minimal, SMLL^0 is actually an interesting and especially well behaved system; we will study its specific properties.

Structures with Boxes. We use boxes to represent the rule for $\bot$:

$$\frac{\vdash \Gamma}{\vdash \bot, \Gamma}$$

The definition of structures with boxes which we adopt is standard. *In short*, a box of conclusions $\bot, \Gamma$ contains a structure R of conclusions Γ, and a distinguished bot link of conclusion $\bot$. Such a conclusion $\bot$ is the *lock* of the box, and R is its *content*. We represent a box graphically as in Figure 3(d), where the structure R is represented as a circle inside the box, and the barred edge labelled by Γ stands for a sequence of edges (the conclusions Γ).

More formally, an SMLL *structure with boxes* is an SMLL structure together with a function which associates to each node l of sort bot a sub-structure R of conclusions Γ (as mentioned above, l and R are depicted in a box). Boxes are required to be either one included in the other, or disjoint. The *depth* of a node is the number of boxes to which it belongs, while the depth of a structure is the maximal depth of its nodes. The lock of a box acts as a guard, as will be evident when we define normalization and the SIAM.

3.2 Correctness

A *net* is a structure (with boxes) which fulfills a correctness criterion. We define correctness by means of switching paths (see [18]). A *switching path* on the structure R is an undirected path[1] which uses:

- for each $\parr$ link, at most one of the two *premisses*;
- for each sync link, at most one of its *out-edges*.

The former condition is standard, the latter condition rules out paths such as the one going from P_1 (below) to N_1 (above) or to P_K (below) in Figure 3(b).

Let us first define correctness for SMLL^0, as it is as immediate as for MLL: an SMLL^0 structure is *correct* if none of its switching paths is cyclic. Correctness for an SMLL structure R is defined by levels, as usual with boxes.

We call *0-graph* of R the restriction to depth 0 of the graph which is obtained from R by replacing each box of conclusion $\bot, \Gamma$ with a new sort of node, labelled as box, which has the same conclusions $\bot, \Gamma$ (like ax, a box node has no premisses). An SMLL structure with boxes is correct — and is said to be an SMLL *net* — if the following conditions hold:

1. there is no switching cycle in the 0-graph of R;
2. the structure inside each box is itself correct.

It is immediate to verify that if we only consider MLL links, we simply have a formulation in terms of switching paths of the usual "acyclicity condition" in the Danos-Regnier criterion[2].

The correctness criterion is a key ingredient to guarantee that synchronizations behave well, *i.e.*, that there are no deadlocks, neither in the normalization nor in the SIAM machine.

[1] By path, in this paper we always mean a *simple path* (no repetition of either nodes or edges).

[2] The Danos-Regnier criterion [8] is actually made of two conditions namely "acyclicity" and "connectedness"; connectedness' only role, however, is to rule out the "mix" rule from the sequent calculus. This is not relevant in our development, so we will ignore it (if wished, one can introduce in the standard way also a connectedness condition; we would then speak of *connected* nets).

Absence of Deadlocks. As already discussed, the central issue associated to the introduction of synchronizations is the need to guarantee the absence of deadlocks, both in the normalization of the nets, and in the runs of the SIAM. We now introduce some technical notions and give a lemma which will be our main tool in all proofs of deadlock freedom. It is common to verify deadlock freedom by using a notion of strict partial order, and this is the case also in our setting. More precisely, we first define a relation on the sync links; the relation corresponds to a notion of dependency that will become clear when we define the SIAM machine. We prove that the relation is a strict partial order; this indicates that there is always at least one sync link which does not depend on any other one.

Given two links l_1, l_2 of an SMLL net, we write $l_1 \prec l_2$ (and we say that l_1 is *before* l_2) if there is a *polarized path* from l_1 to l_2, *i.e.*, a path of polarized edges (connecting polarized nodes) which is going *upwards on negative* edges, and *downwards on positive* edges. We ask that a polarized path does not enter boxes.

LEMMA 3.1 (Links Strict Order). *Given a net R, the set of its links equipped with the relation $\prec$ is a finite strict partial order.*

The result follows from the fact that a polarized path p is in particular a switching path (as one can easily check, noticing that if p crosses a sync link, it uses at most one out-edge); hence a polarized path is never cyclic, and the relation is irreflexive. As a consequence, configurations like the ones in Figure 2 are not possible.

3.3 Normalization

We define a set of rewriting rules on SMLL nets. The elementary reduction steps are given in Figure 4. Reduction is intended to happen at level 0, *i.e.*, reduction *cannot* take place inside boxes. This way we obtain a rewrite relation on structures, called $\to$. It

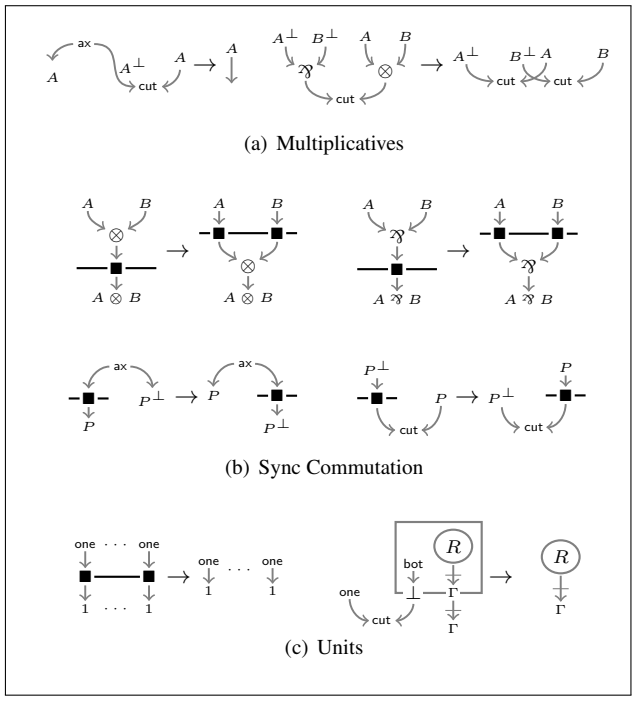


Figure 4. Reduction.

is important to notice that (i) there are no commutations between sync links and that (ii) there are no commutations with a box.

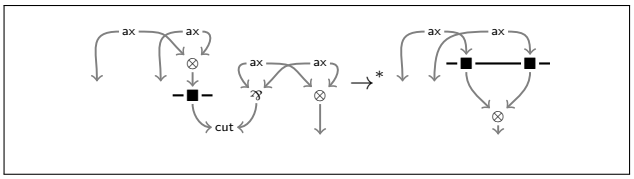


Figure 5. Use of $sync/\otimes$.

REMARK 3.2 (A Minimalistic Alternative). *Several variations on the proposed rewrite rules are indeed possible. Here we have made a choice of* generality*, and of positive-negative* symmetry*. If one aims at obtaining a minimalistic system with all the good properties (in particular, confluence and cut-elimination), one can also choose to apply sync links only to positive formulas. If we assume that all axioms are atomic (*i.e.*, if we make a hypothesis of η-expansion on the axioms), it turns out that instead of the four rules in Figure 4(b), we only need the first one (the $sync/\otimes$ reduction).*

It is now time to study the main properties of the relation $\rightarrow$. The following has the flavour of Subject Reduction, where correctness plays the role of typability:

LEMMA 3.3 (Preservation of Correctness). *If R is a correct structure and $R \rightarrow Q$, then Q is correct, too.*

By standard arguments, and by Lemma 3.1, one can prove that no infinite sequence of reduction can be built:

PROPOSITION 3.4 (Normalization). *The relation $\rightarrow$ is strongly normalizing.*

In principle, many rewrite rules can be applied to a given structure. However, this form of nondeterminism is harmless:

PROPOSITION 3.5 (Confluence). *The relation $\rightarrow$ is confluent.*

The last two results imply that the normal form of any SMLL structure *exists and is unique*. This, by itself, does not mean that cuts can be eliminated, but is an essentially step towards it.

3.4 Cut-Elimination

We now study cut-elimination, *i.e.*, the property that nets in normal form contain no cuts. SMLL^0 turns out to have especially good properties in this respect.

We say that a cut link at depth 0 is *ready* if neither of its premisses is hereditary conclusion of a box. By a straightforward case analysis, we can prove that:

LEMMA 3.6. *Let R be an SMLL net. If there is a ready cut, then R is not a normal form.*

It follows immediately that SMLL^0 enjoys cut-elimination:

THEOREM 3.7 (SMLL^0 Normal Forms). *If R is an SMLL^0 net in normal form, then R is* cut free*.*

We now turn our attention to the whole SMLL. While SMLL^0 enjoys cut-elimination with no conditions, in the presence of boxes we restrict our attention to the *closed* case, *i.e.*, the case in which no $\bot$ appears in the conclusions. We observe that the reduction rules define a *lazy* cut-elimination procedure, because there are no commutations with a box. In the closed case, lazy cut-elimination is enough to eliminate all cuts. The proof makes essential use of Lemma 3.1, together with an adaptation of Girard's analogous result for multiplicative-additive proof-nets [13], which in our setting can be reformulated as follows:

LEMMA 3.8 (Lazy Cut-Elimination). *Let R be a closed SMLL net. If R is normal, then R is cut-free.*

The proof is rather technical, and we refer the reader to the extended version of this paper [5]. As a matter of fact, Lemma 3.8 is a key step towards getting some useful information on the shape of normal forms:

THEOREM 3.9 (SMLL Closed Normal Forms). *The normal forms of closed SMLL nets contain no cuts, no boxes, and no sync links.*

In other words, the normal form of an SMLL net is nothing more than a MLL net!

4. SIAM: an Interactive Model with Synchronizations

The SIAM is a *multi-token* machine designed to run on nets of SMLL. Let us first recall the main features of the IAM, *i.e.*, the standard Interaction Abstract Machine [9, 20]. Given a net R, the IAM pushes a *single* token around R. To each edge of R is associated an *action* — a *transition* — which gives instructions on how to move the token. A *state* of the machine is a position of the token in the net.

To define the SIAM machine M_R associated to a net R, we first need to precisely define what an occurrence of atom and a position are. We can then define the states and the transitions of the SIAM and study its properties.

Occurrences of Atoms. We indicate the occurrences of atoms in a formula by their path in the formula tree. Given an occurrence of atom α in a formula A, its address $\mathtt{addr}_\alpha(A)$ in A is defined as a string on the alphabet $\{\mathbf{l}, \mathbf{r}\}$, by induction:

- if $A = \alpha$, $\mathtt{addr}_\alpha(A) = \varepsilon$;
- if α is in A, then $\mathtt{addr}_\alpha(A * B) = \mathbf{l} \cdot \mathtt{addr}_\alpha(A)$ while $\mathtt{addr}_\alpha(B * A) = \mathbf{r} \cdot \mathtt{addr}_\alpha(A)$, where $*$ is either $\otimes$ or $⅋$.

With a slight abuse of notation, we sometimes identify occurrences of atoms and their address. We indicate addresses with metavariables like m.

Positions. Intuitively, each edge in a net is associated to possibly many positions, one for each occurrence of an atom in the formula A typing it. Given a net R, we define the set of its *positions* $\mathtt{POS}(R)$ as the set including:

- the set of pairs (e, m), where e is an edge of R, and m is (the address of) an occurrence of atom in the formula A typing e;
- together with the set of pairs (e, i), where the edge e is the lock of a box in R, and $i \in \mathbb{N}$. The role played by $i \in \mathbb{N}$ will be explained in Section 4.3.

We say that a position (e, m) is positive or negative if it is the case for the occurrence of atom corresponding to m. We use the metavariables $\mathbf{s}, \mathbf{p}$ to indicate positions. The following subsets of $\mathtt{POS}(R)$ play a crucial role in the following:

- the set $\mathtt{INIT}(R)$ of *initial positions*, that are the negative occurrences of atoms in the conclusions of R;
- the set $\mathtt{FIN}(R)$ of *final positions*, that are the positive occurrences of atoms in the conclusions of R;
- the set $\mathtt{ONES}(R)$ of *one positions*, which are occurrences of 1 which are conclusions of one links;
- the set $\mathtt{BOTBOX}(R)$ of pairs (e, i), where e is the lock of a box.

4.1 The Machine, Formally

It is now time to formally define the SIAM. Given an SMLL net R, the multi-token machine M_R consists in a set of states and a transition relation between them. A state of M_R, intuitively, tells us *how many* tokens currently circulate in the net, and which *positions* they have reached from the initial state, while keeping track (for each of them) of their *origin*.

States. A *state* of M_R is a function $\mathbf{T} : \mathtt{INIT}(R) \cup L \rightarrow \mathtt{POS}(R)$ where $L \subseteq \mathtt{ONES}(R)$. A state is *initial* if $\mathbf{T}$ is the identity on

$\texttt{INIT}(R)$. We indicate the (unique) initial state of R by $\mathbf{I}_R$. A state is *final* if the image of $\mathbf{T}$ is $\texttt{FIN}(R) \cup B$, with $B \subseteq \texttt{BOTBOX}(R)$. Intuitively, we identify each token with its original position $\mathbf{s}$; given a state $\mathbf{T}$ of M_R, we say that *there is* a token on the position $\mathbf{p}$ if $\mathbf{T}(\mathbf{s}) = \mathbf{p}$, for $\mathbf{s} \in dom(\mathbf{T})$. We use expressions such as "a token moves" or "crossing a link" in the intuitive way. We will refer to the set of all tokens as *the multi-token*.

In describing M_R we also use the following notions: an edge e of type A is said to be *saturated* if each position of e is in the image of $\mathbf{T}$. A box is *unlocked* if e is the lock of the box, and $(e, i) \in range(\mathbf{T})$. A link is *active*, if it is either at depth 0 in R, or at depth 0 inside an unlocked box.

Transitions. The *transition rules* are in Figure 6 (where $*$ stands for either $\parr$ or $\otimes$). To represent the position $\mathbf{p} = (e, m)$ (respectively, $\mathbf{p} = (e, i)$, $i \in \mathbb{N}$) we write a bullet $\bullet$ along e, together with the atom occurrence m (respectively, the value i) next to it. The symbol $\downarrow$ (respectively, $\uparrow$) is a polarity annotation: it indicates that the position is positive (respectively, negative). To represent a transition $\mathbf{T} \longrightarrow \mathbf{U}$, we depict $\mathbf{T}(\mathbf{s})$ in the left-hand-side, and $\mathbf{U}(\mathbf{s})$ on the right-hand-side of the arrow, for each $\mathbf{s} \in dom(\mathbf{T})$ such that $\mathbf{U}(\mathbf{s}) \neq \mathbf{T}(\mathbf{s})$. It is of course intended that $\mathbf{U}(\mathbf{s}) = \mathbf{T}(\mathbf{s})$ for all $\mathbf{s}$ whose value is not explicitly appearing in the picture.

Observe that the (positive or negative) polarity of any position determines its direction: a token on a negative atom always moves upwards, while a token on a positive atom always moves downwards. This is coherent with the way initial positions are defined.

1. ax, cut, $\otimes$, $\parr$: the transitions are the same as in MLL;
2. *Synchronization*. Tokens cross a sync link l only when each in-edge of l is saturated (in the pictures, if e is an edge of type A, we write $\overline{m}_A$ for the set of all the occurrence of atom of A); all tokens cross the link simultaneously.
3. The transition associated to a one link of conclusion e has two conditions: (i) the one link needs to be active, (ii) $\mathbf{p} = (e, \varepsilon) \notin dom(\mathbf{T})$. In this case, $\mathbf{T}$ is extended with the identity on the position $\mathbf{p}$. This is the only transition changing the domain of $\mathbf{T}$.
4. *Boxes.* When the token goes through the conclusion of a box (graphically, the token "crosses" the border of the box), it is modified as if it were crossing a node:
   i. If there is a token on the lock e of a box (necessarily $m = \varepsilon$), then (e, ε) becomes $(e, 0) \in \texttt{BOTBOX}(R)$. This transition plays no significant role at the moment, but we clarify its role in Section 4.3, and we will make essential use of it in Section 6.
   ii. Tokens can enter a box only when the box is unlocked. As a consequence, if a box is not unlocked, then no token can be inside the box.

Why Polarized Formulas Only? If we synchronize the two conclusions $1, \bot$ of an atomic axiom, as depicted in Figure 7(a), what we obtain is a deadlocked structure: the sync link needs to have a token at the same time on the in-edge of type $\bot$, and on the in-edge of type 1, which is not possible, since that would be the *same* token. On the other hand, this configuration is ruled out by the correctness criterion: the structure at hand has a switching cycle. However, if we join the two atoms with a $\parr$, and apply the sync link on $1 \parr \bot$, the situation does not change, but the criterion does not catch the deadlock; moreover, after a step of reduction, a cycle would appear (see Figure 7(b)).

4.2 SIAM: Properties and Soundness

In this section, we study the properties of the SIAM and in particular termination, confluence, and deadlock freedom. We also show that the SIAM is (up to equivalence) a model of cut-elimination

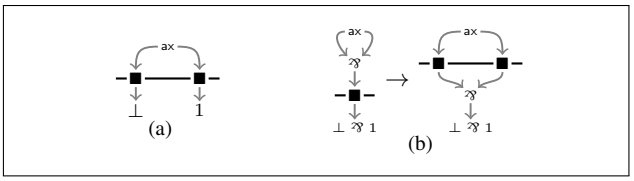


Figure 7. The Need for Polarization.

for SMLL, *i.e.*, a Soundness Theorem. All along this section, R indicates an SMLL net, M_R its multi-token machine, and $\longrightarrow$ the induced relation. Let us first establish some basic properties of $\longrightarrow$.

LEMMA 4.1. *1. Each* $\mathbf{T}$ *such that* $\mathbf{I}_R \longrightarrow^* \mathbf{T}$ *is an* injective *function.*
2. There are no infinite sequences of transitions from $\mathbf{I}_R$.
3. $\longrightarrow$ *is* confluent.

A *run* of the SIAM machine on R is a maximal sequence $\mathbf{I}_R \longrightarrow \mathbf{T}_1 \longrightarrow \mathbf{T}_2 \longrightarrow ... \longrightarrow \mathbf{T}_n$ of transitions from the initial state $\mathbf{I}_R$. The lemma above guarantees that *each run of the machine* M_R *terminates*, and that the normal form $\mathbf{T}_n$ reached from the initial state exists and is unique.

A central property which we still need to prove is *deadlock freedom*, *i.e.*, if $\mathbf{I}_R \longrightarrow^* \mathbf{T}$, and no reduction applies, then $\mathbf{T}$ is a final state. By confluence, and hence unicity of the normal form, we know that if there is a run of M_R which terminates on a final state $\mathbf{F}$, then all runs of the machine terminate on $\mathbf{F}$. Deadlock freedom will be a consequence of *soundness*, namely that if R reduces to S then M_R is somehow equivalent to M_S. To make this precise, we define the interpretation $[R]$ of a net R as:

$$[R] : (\texttt{INIT}(R) \rightharpoonup \texttt{FIN}(R)) \cup \{\Box\}$$

We remind that $\texttt{INIT}(R)$ and $\texttt{FIN}(R)$ are the initial and final positions. The formal symbol $\Box$ indicates that the machine M_R deadlocks when starting in its (unique!) initial state. Otherwise, $\mathbf{I}_R \longrightarrow^* \mathbf{F}$, where $\mathbf{F}$ is a final state, and $[R]$ is defined to be the partial function obtained as the restriction of $\mathbf{F}$ to the elements of $\texttt{INIT}(R)$ whose image is not in $\texttt{BOTBOX}(R)$.

THEOREM 4.2 (Soundness). *If* R *is a net, and* S *is its normal form, then* $[R] = [S]$.

Proof. Let R be a net of conclusions $A_1, \ldots, A_n$, and $R \longrightarrow S$; to be able to compare R and S, we adopt the following convention: we identify each conclusion of a net with the occurrence of formula A_i typing it, so that there is no ambiguity. In particular, R and S have the same initial and final positions. We now show that the interpretation of a net is preserved by all normalization steps; in particular, if $R \longrightarrow S$, then M_R deadlocks if and only if M_S deadlocks. We look at the reductions in Figure 4. If the multi-token never reaches the portion of the net which is involved in the reduction, then both $[R]$ and $[S]$ have the same value. Otherwise, we analyze all cases, and verify that there is a deadlock in M_R iff there is a deadlock in M_S, and otherwise, a run of the machine continues in a similar way, so that $[R] = [S]$. Let us just examine a few cases:

- *Sync Elimination*. In both the l.h.s and the r.h.s. all one links are active, because they are at level 0. Thus, we are able to have a token on the conclusion of each one link, and after one step, on each conclusion of the sync link.
- *Box Opening*. The one link l on the l.h.s. is active; we can therefore add the position $\mathbf{p}$ associated with l to the domain of the state. We now have a token on $\mathbf{p}$; the token crosses the cut, and unlocks the box. We observe that $\mathbf{p}$ does not belong to the domain of $[R]$. Once the box is unlocked, the SIAM behaves exactly in the same way on the l.h.s. and the r.h.s.: any token

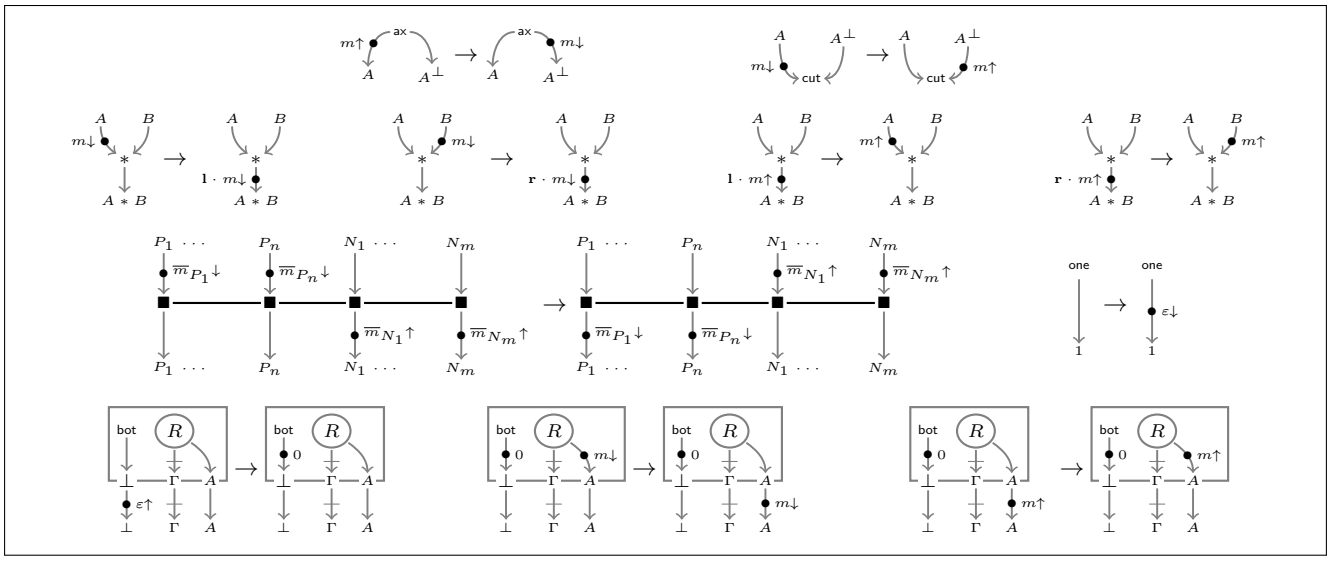


Figure 6. SIAM Transition Rules.

reaching a position in Γ will continue in exactly the same way in both sides.

This concludes the proof. □

A consequence of Theorem 4.2 is that M_R deadlocks iff M_S deadlocks. Soundness, thus, allows us to study deadlock freedom of the SIAM by studying deadlock freedom of the SIAM when running on a cut-free net.

THEOREM 4.3 (Termination and Deadlock Freedom). *Let R be a net, and M_R its multi-token machine. All runs of M_R terminate on a final state. Moreover, such a final state is unique.*

This result holds for both SMLL^0 and SMLL nets. However, the argument is quite different, because the results which we have on cut-elimination are themselves different:

1. In the case of SMLL^0, we can exploit the fact that the normal form of a net is cut-free (Theorem 3.7). The difficulty comes from the fact that in the normal form we do have sync links. However, if a net is cut-free, all sync links are hereditary conclusions of ax links, and the correctness criterion (via Lemma 3.1) allows us to establish that the tokens cannot get stuck before reaching a final state. Together with soundness, this allows us to conclude.
2. In the case of SMLL, on the one hand the proof is simplified by the fact that all sync links can be eliminated, but on the other hand, we have a result of cut-elimination which is limited to the closed case. We modularize the proof into two steps:
   i. if R is an SMLL net with no $\bot$ in the conclusion, Theorem 4.3 is an immediate consequence of Soundness and Theorem 3.9;
   ii. we then prove the result for an arbitrary SMLL net R by providing a construction that embeds R into a larger net $\widehat{R}$ (its closure) which has no $\bot$ in its conclusion (therefore Point 1. applies), and by showing that $M_{\widehat{R}}$ deadlocks iff M_R deadlocks.

One may wonder what happens to the sync links in case (2.i). Normalization pushes the sync's upwards, and since we are in the closed case, eventually all sync links are hereditary conclusions of one links. The elimination of the sync links (Theorem 3.9) can thus be seen as a step towards Deadlock Freedom: Lemma 3.1 guarantees that no situation like the one in Figure 2(b) can arise (thus, there is always a top-most sync link which can be eliminated).

4.3 Multi-Boxes

We have everything in place to model also probabilistic or nondeterministic choice, with only a small modification of SMLL, where the content of a box can be not a *single* net, but a *sequence* of $n > 1$ nets. A *multi-box* of conclusion $\bot, \Gamma$ may contain several nets $R_1, \ldots, R_n$, all of the same conclusion Γ (see Figure 8). In

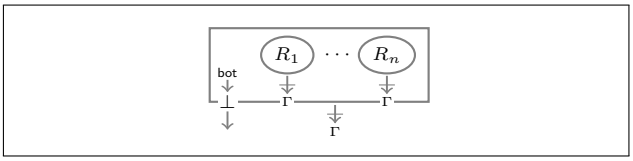


Figure 8. A Multi-box.

sequent calculus, this would correspond to the rule

$$\frac{\vdash \Gamma \quad \cdots \quad \vdash \Gamma}{\vdash \bot, \Gamma}$$

The normalization relation becomes nondeterministic, as it chooses one of the nets inside the box when opening it. All the interesting properties hold, with very similar proofs.

The SIAM needs to be adapted slightly, but in fact was already set for this (see Point 4 in the discussion on transitions). If a box contains n nets, and a token reaches the lock of the box, the machine choose a value $i \in \{0, \ldots, n-1\}$ nondeterministically; the integer i now determines to which of the R_i internal nets the incoming tokens will move (only R_i is active). As a consequence, for any box, at most one of the internal nets is populated and traversed by tokens. If we call *slice* a choice of a single subnet for each box, one can check that a run of the machine always happens in a single slice, and has therefore properties similar to the ones we studied above. We will put this at work in the following section.

$$\frac{}{x : A \vdash x : A} \qquad \frac{}{r \vdash r : \mathbb{Q}} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}$$

$$\frac{\Gamma \vdash t : A \multimap B \qquad \Delta \vdash u : A}{\Gamma, \Delta \vdash tu : B} \qquad \frac{\Gamma \vdash t : A \qquad \Delta \vdash u : B}{\Gamma, \Delta \vdash \langle t, u \rangle : A \otimes B}$$

$$\frac{\Gamma \vdash t : A \otimes B \qquad \Delta, x : A, y : B \vdash u : C}{\Gamma, \Delta \vdash \mathsf{let}\ \langle x, y \rangle\ \mathsf{be}\ t\ \mathsf{in}\ u : C}$$

$$\frac{}{\cdot \vdash \mathsf{tt} : \mathbb{B}} \qquad \frac{}{\cdot \vdash \mathsf{ff} : \mathbb{B}} \qquad \frac{\Gamma \vdash t : \mathbb{B} \qquad \vdash u : A \qquad \vdash v : A}{\Gamma \vdash \mathsf{if}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v : A}$$

$$\frac{}{\cdot \vdash \mathsf{new} : \mathbb{Q}} \qquad \frac{}{\cdot \vdash \mathsf{U}_n : \mathbb{Q}^{\otimes n} \multimap \mathbb{Q}^{\otimes n}} \qquad \frac{}{\cdot \vdash \mathsf{meas} : \mathbb{Q} \multimap \mathbb{B}}$$

Figure 9. Typing Rules.

5. Synchronization, Proof-nets, and Quantum Computation

In this section, the whole development of the last two sections will be applied to higher-order quantum computation in the form of a linear quantum λ-calculus akin to that recently introduced by Selinger and Valiron [24].

5.1 A Linear Quantum λ-calculus

We assume given a finite set $\mathcal{UO}$ of symbols, each denoting a unitary operator. This is ranged over by metavariables like U and V. When we want to insist on U having arity n, we write it as U_n. *Terms* are defined as follows:

$$t, u, v ::= x \mid r \mid tu \mid \lambda x.t \mid \langle t, u \rangle \mid \mathsf{let}\ \langle x, y \rangle\ \mathsf{be}\ t\ \mathsf{in}\ u \mid \mathsf{new} \mid \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{meas} \mid \mathsf{U} \mid \mathsf{if}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v,$$

where r ranges in a denumerable set $\mathcal{QV}$ of quantum variables. *Types* are defined as follows:

$$A, B ::= \mathbb{B} \mid \mathbb{Q} \mid A \multimap B \mid A \otimes B.$$

For every natural number n and for every type A, the expression $A^{\otimes n}$ stands for the type $A \otimes \cdots \otimes A$ (n times). *Typing judgments* are in the form $\Theta, V \vdash t : A$ where t is a term, A is a type, Θ is an environment mapping λ-variables to types, and $V \subseteq \mathcal{QV}$. Expressions like Θ, V are called *contexts* and are denoted by metavariables like Γ or Δ. *Typing rules* are as in Figure 9. Observe how terms in the form $\mathsf{if}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v$ are typed in a slightly non-standard way, in that u and v are required to be *closed*. In presence of higher-order types and nameless functions, however, this does not cause any significant threat to expressivity. If a derivation π has conclusion $\Gamma \vdash t : A$, then we write $\pi \triangleright \Gamma \vdash t : A$. *Values* are terms generated by the following grammar:

$$a, b ::= r \mid \lambda x.t \mid \langle a, b \rangle \mid \mathsf{new} \mid \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{meas} \mid \mathsf{U}.$$

Evaluation contexts are expressions built as follows:

$$E, F, G ::= [\cdot] \mid Et \mid aF \mid \langle E, t \rangle \mid \langle a, F \rangle \mid \mathsf{let}\ \langle x, y \rangle\ \mathsf{be}\ E\ \mathsf{in}\ t \mid \mathsf{if}\ E\ \mathsf{then}\ t\ \mathsf{else}\ u.$$

As usual, we indicate as $E[t]$ the term obtained by filling the only occurrence of $[\cdot]$ in E with t.

5.2 Quantum Closures and Operational Semantics

Given any finite set X, $\mathbb{H}(X)$ is the finite-dimensional Hilbert space generated by X, *i.e.*, the Hilbert Space having X as basis. Let X be any finite set. By $\mathcal{SUB}(X)$ we note the (finite) set of all functions mapping elements of X to bits in $\{0, 1\}$. Given an element Q of $\mathbb{H}(\mathcal{SUB}(X))$, an element $r \in X$ and a bit $b \in \{0, 1\}$, we indicate with $\mathcal{J}_b^r(Q)$ the projection of Q into the subspace of $\mathbb{H}(\mathcal{SUB}(X))$ in which r is assigned to b. Similarly, $\mathcal{R}_b^r(Q)$ is the probability of observing b when measuring the value of r in Q.

A *quantum closure* of type A is a pair $[Q, t]$, where Q is a normalized vector in $\mathbb{H}(\mathcal{SUB}(V))$ and $V \vdash t : A$. In this case we often write $[Q, t] : A$. Quantum closures are taken modulo a form of α-equivalence in which one is allowed to change the name of a quantum variable, modifying the underlying quantum register accordingly. Quantum closures are denoted with metavariables like C and D. $\mathcal{C}_A$ is the set of all quantum closures of type A, while $\mathcal{C}$ is the set of all quantum closures.

We need to work with distributions on quantum closures, namely functions in the form $\mathscr{D} : \mathcal{C} \to \mathbb{R}_{[0,1]}$ satisfying $\sum_{C \in \mathcal{C}} \mathscr{D}(C) \leq 1$. The subset of $\mathcal{C}$ of those quantum closures C for which $\mathscr{D}(C) > 0$ is said to be the *support* of $\mathscr{D}$ and is denoted as $\mathsf{S}(\mathscr{D})$. In the following, we will only be concerned with distributions having (at most) denumerable support. Distributions having a finite support are indicated with expressions like $\{C_1^{p_1}, \ldots, C_n^{p_n}\}$, with the obvious meaning. The set of distributions over a set X is denoted as $\mathbb{D}(X)$.

Quantum closures can be given a semantics in two steps: first, one gives a binary relation $\mapsto$ between quantum closures and distributions capturing one-step reduction, then one generalizes the relation above to another relation $\Rightarrow$ capturing multi-step reduction. Rules for $\mapsto$ are in Figure 10. Reduction along $\mapsto$ preserves types:

LEMMA 5.1 (Subject Reduction). *If $C : A$ and $C \mapsto \mathscr{D}$, then $D : A$ for every $D \in \mathsf{S}(\mathscr{D})$.*

A quantum closure C is in *normal form* if for any distribution $\mathscr{D}$ it does not hold that $C \mapsto \mathscr{D}$. Reduction cannot get stuck, as expected:

LEMMA 5.2 (Progress). *If $[Q, t] : A$ is a quantum closure in normal form, then t is a value.*

The multistep relation $\Rightarrow$ is defined by the following set of rules:

$$\frac{}{C \Rightarrow \{C^1\}} \qquad \frac{C \mapsto \{D_1^{p_1}, \ldots, D_n^{p_n}\} \qquad D_i \Rightarrow \mathscr{D}_i}{C \Rightarrow \sum_{i=1}^n p_i \cdot \mathscr{D}_i}$$

5.3 Translation into QSMLL

Type derivations of the λ-calculus which we have just introduced will be mapped into nets for an immediate generalization of SMLL, called QSMLL. Specifically, we need to generalize SMLL in the following two ways:

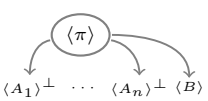
- *Synchronization nodes are labelled with a unitary operator* whose arity is the sum of the number of atom occurrences in the involved formulas.
- *Boxes contain two nets.*

Types can be translated into formulas as follows:

$$\langle \mathbb{B} \rangle = \langle \mathbb{Q} \rangle = 1; \qquad \langle A \multimap B \rangle = \langle A \rangle^{\perp} \parr \langle B \rangle;$$

$$\langle A \otimes B \rangle = \langle A \rangle \otimes \langle B \rangle.$$

Any type derivation π with conclusions $x_1 : A_1, \ldots, x_n : A_n, r_1, \ldots, r_m \vdash t : B$ can be translated into a net $\langle \pi \rangle$ of the following shape:



The construction is an induction on the structure of π, see [5] for more details.

Extending the Translation to Quantum Closures. The translation scheme above can be extended to quantum closures, again by gen-

$$\frac{}{[Q, E[(\lambda x.t)u]] \mapsto \{[Q, E[t\{u/x\}]]^1\}} \qquad \frac{}{[Q, E[\mathsf{let}\ \langle x, y\rangle\ \mathsf{be}\ \langle t, u\rangle\ \mathsf{in}\ v]] \mapsto \{[Q, E[v\{t, u/x, y\}]]^1\}}$$

$$\frac{}{[Q, E[\mathsf{if}\ \mathsf{tt}\ \mathsf{then}\ t\ \mathsf{else}\ u]] \mapsto \{[Q, E[t]]^1\}} \qquad \frac{}{[Q, E[\mathsf{if}\ \mathsf{ff}\ \mathsf{then}\ t\ \mathsf{else}\ u]] \mapsto \{[Q, E[u]]^1\}}$$

$$\frac{r\ \text{is fresh}}{[Q, E[\mathsf{new}]] \mapsto [Q \otimes |r \leftarrow 0\rangle, E[r]]} \qquad \frac{}{[Q, E[\mathsf{U}_n\langle r_1, \ldots, r_n\rangle]] \mapsto \{[\mathsf{U}_n^{x_1,\ldots,x_n}(Q), E[\langle r_1, \ldots, r_n\rangle]]^1\}}$$

$$\frac{}{[Q, E[\mathsf{meas}\ r]] \mapsto \{[\mathcal{J}_0^r(Q), E[\mathsf{tt}]]^{\mathcal{R}_0^r(Q)}, [\mathcal{J}_1^r(Q), E[\mathsf{ff}]]^{\mathcal{R}_1^r(Q)}\}}$$

Figure 10. Operational Semantics.

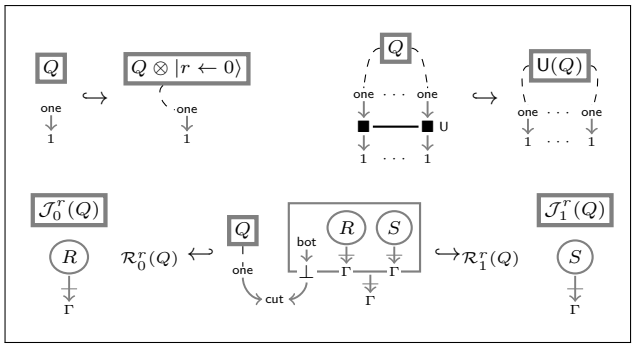
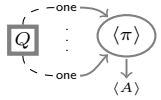


Figure 11. Reduction — New Rules.

eralizing the notion of net; to a net we now associate a quantum register, as follows. A *quantum net* is a pair (R, Q), where:

- R is a QSMLL net;
- Q is a normalized vector in $\mathbb{H}(\{l_1, \ldots, l_n\})$, where $l_1, \ldots, l_n$ are one links of R which are at depth 0.

The generalization above allows to map any quantum closure $[Q, t]$ into the following quantum net:



where π is a type derivation for t. Please observe the way we explicit the correspondence between (some of the) one links and the quantum register Q by a dashed line. This way, any quantum closure C can be associated to a quantum net $\langle C\rangle$, and the map $\langle\cdot\rangle$ can be generalized to a function mapping distributions of quantum closures to distributions of quantum nets.

5.4 Normalization

The additional features of QSMLL require an adaptation of the cut-elimination procedure. Moreover, reduction becomes probabilistic. We are then forced to consider distributions over nets, which we note $\mathscr{R}, \mathscr{Q}, \ldots$, in the same way as the distributions over quantum closures we considered in Section 5.2. The new reduction rules are the ones in Figure 11. Analogously to what happens for λ-terms, then, two reduction relations can be defined on nets, namely a one-step reduction relation $\hookrightarrow$ and a multi-step reduction relation $\rightsquigarrow$, both between nets and distributions over nets. The main result of this section is the following:

PROPOSITION 5.3 (Simulation). *If $C \Rightarrow \mathscr{D}$, then $\langle C\rangle \rightsquigarrow \langle\mathscr{D}\rangle$.*

The main technical lemma towards Proposition 5.3 is the following Substitution Lemma, where $\mathsf{cut}(\pi, x.\psi)$ is the net obtained by $\langle\pi\rangle$ and $\langle\psi\rangle$, cutting the link for x in the former to the conclusion for the latter.

LEMMA 5.4 (Subsitution Lemma). *Suppose $\pi \triangleright x : A, r_1, \ldots, r_n \vdash t : B$ and $\psi \triangleright q_1, \ldots, q_m \vdash u : A$. Then $\mathsf{cut}(\pi, x.\psi) \rightsquigarrow \{\langle\pi\{\psi/x\}\rangle^1\}$.*

6. Computing with the QSIAM

The SIAM can be generalized to the QSIAM, an abstract machine for quantum nets. Most definitions about the QSIAM are inherited from those about the SIAM (see [5] for more details). In particular, positions and sets of positions such as $\mathtt{POS}(R)$, $\mathtt{INIT}(R)$, etc. are defined exactly in the same way.

Statics. The states of $M_{\mathbf{R}}$, the interactive machine interpreting the quantum net $\mathbf{R} = (R, Q)$, are the pairs $s = (\mathbf{T}, P)$ where:

- $\mathbf{T}$ is a function from $\mathtt{INIT}(R) \cup L$ to $\mathtt{POS}(R)$, where L is a subset of $\mathtt{ONES}(R)$. As in the SIAM, the role of $\mathbf{T}$ is to capture *where* the $|L|$ tokens started from (namely the positions in L) and where they are (namely the positions in $\mathbf{T}(L)$).
- P is a quantum register in $\mathbb{H}(\mathcal{SUB}(L))$.

The set of states of $M_{\mathbf{R}}$ is denoted by $ST(\mathbf{R})$. The notions of an initial state, a final state, and an active state are very close to the ones given for the SIAM. The set of all final states of $M_{\mathbf{R}}$ is indicated by $FST(\mathbf{R})$.

Dynamics. It is now time to describe how states evolve. This takes again the form of a relation $\hookrightarrow$ between $ST(\mathbf{R})$ and *finite distributions* over the same set. Rules follow quite closely the ones for the SIAM and those which are peculiar to the QSIAM can be found in [5]. Extending $\hookrightarrow$ into a multi-step relation $\twoheadrightarrow$ can be done in the usual, standard, way:

$$\frac{}{s \twoheadrightarrow \emptyset} \qquad \frac{s \in FST(\mathbf{R})}{s \twoheadrightarrow \{s^1\}} \qquad \frac{s \hookrightarrow \{t_1^{p_1}, \ldots, t_n^{p_n}\} \qquad t_i \twoheadrightarrow \mathscr{D}_i}{s \twoheadrightarrow \sum_{i=1}^n p_i \cdot \mathscr{D}_i}$$

The semantics of any state s is simply

$$[\![s]\!] = \sup_{s \twoheadrightarrow \mathscr{D}} \mathscr{D}.$$

Analogously to what has been done for SMLL, we can define the object $[\mathbf{R}]$ computed by the QSIAM for $\mathbf{R}$. We can then prove the following:

THEOREM 6.1 (Soundness). *If $\mathbf{R} \rightsquigarrow \mathscr{R}$, then $[\mathbf{R}] = [\mathscr{R}]$.*

Combining Theorem 6.1 and Proposition 5.3, one gets that the QSIAM is a model of computation which is adequate with respect to the operational semantics introduced in Section 5.2:

COROLLARY 6.2. *If $C \Rightarrow \mathscr{D}$, then $[\langle C\rangle] = [\langle\mathscr{D}\rangle]$.*

7. Discussion

Sequentialization. The graphical calculus we propose here does not have a sequent calculus counterpart, at least not a standard one.

One would need to add some extra information (for example, a coherence relation on atoms), to express the fact that atoms from different axioms can be synchronized (or entangled).

A variation on SMLL which would admit sequentialization into a sequent calculus is obtained by forcing all sync links to be *unary*, *i.e.*, to have a single premiss and a single conclusion. To such a link one can easily associate a sequent calculus rule, or a term derivation. However, cut-elimination only holds if the net is closed. Such a solution was explored in [23]. A more sophisticated but somehow similar solution has been proposed by one of the anonymous referees, who suggested sync links to be identified with a new kind of *synchronous* cut.

There is actually a trade-off between two desirable results here: sequentialization and cut-elimination. A good example is the one in Figure 5, which can be understood as the net associated to a term in the form $\mathsf{let}\ \langle z, w\rangle\ \mathsf{be}\ (\mathsf{U}\langle x, y\rangle)\ \mathsf{in}\ \langle z, w\rangle$. The net on the l.h.s. cannot be reduced further if we limit ourselves to unary sink links. On the other hand, to this net we can associate a sequent calculus proof, while it is not the case for the net on the r.h.s. In this paper, we prefer to have cut-elimination without conditions, because cut-elimination gives us a tool to deal with deadlock freedom.

Compiling Terms into Circuits. The QSIAM machine is definitely a quantum automaton: unitary transformations and measurements are performed while visiting the net. It would also be interesting, especially in the measurement-free case, to design token machines which *extract* a quantum circuit from a net instead of executing it on-the-fly. The obtained machine would of course be sound only in the absence of the sync elimination rule, so that in the normal form (which would essentially be a SMLL^0 net) the unitary gates remain explicit. By the way, having this option plays in favor of the choice discussed in the paragraph above, since, again, it is cut-elimination which allows us to prove the absence of deadlocks.

8. Conclusions

This work can be seen as the first step towards making Interaction Abstract Machines a more general model of computation in which not only parallelism, but also synchronization, can take place. Interestingly, this is done with tools coming from proof-theory, namely proof-nets. Noticeably, desirable properties like termination and deadlock freedom are byproduct of correctness.

The main weakness of this work is that the underlying logical system, namely MLL, is of limited expressive power. Adding exponential connectives to SMLL is quite natural, and has not been done here only for the sake of simplicity. Another point worth investigating is certainly a further analysis on the nature of synchronization, and in particular on the possibility of synchronizing over formulas neither strictly positive nor strictly negative. In general, this can lead to deadlocks, but how about isolating a class of *safe* formulas?

Acknowledgments

The first author is supported by the project ANR-12IS02001 "PACE". The second author is supported by the project ANR-2010-BLANC-021301 "LOGOI". The third and fourth authors are supported by Grants-in-Aid for Young Scientists (A) No. 24680001, JSPS, and by Aihara Innovative Mathematical Modeling Project, FIRST Program, JSPS/CSTP.

References

[1] S. Abramsky, E. Haghverdi, and P. J. Scott. Geometry of interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002.

[2] P. Baillot and M. Pedicini. Elementary complexity and geometry of interaction. *Fundam. Inform.*, 45(1-2):1–31, 2001.

[3] P.-L. Curien and H. Herbelin. Abstract machines for dialogue games. Available at `http://arxiv.org/abs/0706.2544`, 2007.

[4] U. Dal Lago. Context semantics, linear logic, and computational complexity. *ACM Trans. Comput. Log.*, 10(4), 2009.

[5] U. Dal Lago, C. Faggian, I. Hasuo, and A. Yoshimitzu. The geometry of synchronization (long version). Available at `http://arxiv.org/abs/1405.3427`, 2014.

[6] U. Dal Lago and M. Zorzi. Wave-style token machines and quantum lambda calculi. Available at `http://arxiv.org/abs/1307.0550`, 2013.

[7] V. Danos and J.-L. Krivine. Disjunctive tautologies as synchronisation schemes. In *CSL*, pages 292–301, 2000.

[8] V. Danos and L. Regnier. The structure of multiplicatives. *Archive for Mathematical logic*, 28(3):181–203, 1989.

[9] V. Danos and L. Regnier. Reversible, irreversible and optimal lambda-machines. *Theor. Comput. Sci.*, 227(1-2):79–97, 1999.

[10] O. Fredriksson and D. R. Ghica. Abstract machines for game semantics, revisited. In *LICS*, pages 560–569, 2013.

[11] J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[12] J.-Y. Girard. Geometry of interaction III: Accomodating the additives. In *Advances in Linear Logic*, number 222 in London Mathematical Society Lecture Notes Series. Cambridge University Press, 1995.

[13] J.-Y. Girard. Proof-nets: The parallel syntax for proof-theory. In *Logic and Algebra*, pages 97–124. Marcel Dekker, 1996.

[14] J.-Y. Girard. Geometry of interaction V: Logic in the hyperfinite factor. *Theor. Comput. Sci.*, 412(20):1860–1883, 2011.

[15] G. Gonthier, M. Abadi, and J.-J. Lévy. Linear logic without boxes. In *LICS*, pages 223–234, 1992.

[16] Y. Hirai. *Hyper-Lambda Calculi*. Phd thesis, University of Tokyo, 2013.

[17] R. Jozsa. Entanglement and quantum computation. Available at `http://arxiv.org/abs/quant-ph/9707034`, 1997.

[18] O. Laurent. An introduction to proof nets. Available at `http://perso.ens-lyon.fr/olivier.laurent/pn.pdf`.

[19] O. Laurent. *Etude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, Mar. 2002.

[20] I. Mackie. The geometry of interaction machine. In *POPL*, pages 198–208, 1995.

[21] M. Pedicini and F. Quaglia. A parallel implementation for optimal lambda-calculus reduction. In *PPDP*, pages 3–14, 2000.

[22] J. S. Pinto. Parallel implementation models for the lambda-calculus using the geometry of interaction. In *TLCA*, pages 385–399, 2001.

[23] T. Roussel. Sémantique opérationnelle d'un lambda calcul quantique via la géométrie de l'interaction. Master's thesis, Université Paris 7, 2012.

[24] P. Selinger and B. Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16(3):527–552, 2006.

[25] K. Terui. Proof nets and boolean circuits. In *LICS*, pages 182–191, 2004.

[26] A. Yoshimizu, I. Hasuo, C. Faggian, and U. Dal Lago. Measurements in proof nets as higher-order quantum circuits. In *ESOP*, pages 371–391, 2014.