A Graph Abstract Machine Describing Event Structure Composition

Claudia Faggian and Mauro Piccolo [1,2,3]

Dipartimento di Matematica Pura e Applicata – PPS
Universitá di Padova – Paris7-CNRS

Abstract

Event structures, Game Semantics strategies and Linear Logic proof-nets arise in different domains (concurrency, semantics, proof-theory) but can all be described by means of directed acyclic graphs (dag's). They are all equipped with a specific notion of composition, interaction or normalization.
We report on-going work, aiming to investigate the common dynamics which seems to underly these different structures.
In this paper we focus on confusion free event structures on one side, and linear strategies [Gir01,FM05] on the other side. We introduce an abstract machine which is based on (and generalizes) strategies interaction; it processes labelled dag's, and provides a common presentation of the composition at work in these different settings.

1 Introduction

Event structures [NPW81], Game Semantics strategies and Linear Logic proof-nets [Gir87] arise in different domains (concurrency, semantics, proof-theory) but can all be described by means of directed acyclic graphs (dag's). They are all equipped with a specific notion of composition, interaction or normalization. In this paper we report ongoing work whose first aim is to investigate the common dynamics which appears to underly all these different structures, and eventually to transfer technologies between these settings.

As a first step in this direction, we present an abstract machine which processes labelled dag's. The machine is based on the dynamics at work when composing Game Semantics strategies. When applied to linear strategies (in the form introduced in [Gir01] or [FM05]) the machine implements strategies composition. When

[1] This work has been motivated from discussion with Nobuko Yoshida and Daniele Varacca.
We are in debt with Daniele Varacca for many explanations, comments, and suggestions.
We are grateful to Martin Hyland, Emmanuel Beffara, and Pierre-Louis Curien for interesting discussions.
We also wish to thank the referees for many usefull remarks and suggestions.

[2] Email: claudia@math.unipd.it

[3] Email: piccolo@di.unito.it

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

applied to event structures, the result is the same as the *paralle composition* of event structures defined by Varacca and Yoshida in [VY06].

Event structures are a *causal model* of concurrency (also called true concurrency models), i.e. causality, concurrency and conflict are directly represented, as opposite to *interleaving models*, which describe the system by means of all possible scheduling of concurrent actions.

An event structure models parallel computation by means of *occurrence of events*, and a partial order expressing *causal dependency*. *Non-determinism* is modelled by means of a *conflict relation*, which expresses how the occurrence of certain events rules out the occurrence of others.

Two events are *concurrent* if they are neither causally related, nor in conflict. Events which are in conflict live in different possible evolutions of the system.

In this paper we will consider two classes of event structures: *confusion free event structure* (where conflict, and hence non-determinism, is well behaving), and *conflict free event structures* (where there is no conflict, and hence no non-determinism).

Confusion free event structure, are an important class of event structures because the choices which a process can do are "local" and not influenced by independent actions. In this sense, confusion freeness generalizes *confluence* to systems that allow nondeterminism.

A point which is central to our approach is that a confusion free event structure $\mathcal{E}$ can be seen as a superposition of conflict-free event structures (which we will call the *slices* of $\mathcal{E}$): each slice represents a possible and *independent* evolution of the system. Because of this, if $\mathcal{E}$ is confusion free, the study of several properties can be reduced to the study of such properties in conflict free event structures.

Games and strategies provide denotational models for programming languages and logical systems; games correspond to types (formulas), and strategies to programs (proofs). The central notion is that of interaction, which models program composition (normalization of proofs).

A distinction between causal and interleaving models is appearing also in Game Semantics. In this setting, a *strategy* describes in an abstract way the operational behaviour of a term. In the standard approach, a strategy is described by sequences of actions (*plays*), which represent the traces of the computation. However, there is an active line of research in Game Semantics aiming at relaxing sequentiality, either with the purpose to have "partial order" models of programming languages or to capture concurrency [AM99,Mel04,HS02,MW05,SPP05,FM05,CF05,Lai05,GM04]. The underlying idea is to not completely specify the order in which the actions should be performed, while still being able to express constraints. Certain tasks may have to be performed before other tasks; other actions can be performed in parallel, or scheduled in any order. A strategy of this kind can be presented as a *directed acyclic graph*.

Content. An idea which underlies the work on typed π-calulus by Honda and Yoshida is that typed processes should be seen as a sort of Hyland-Ong strategies (see for example [NYB01]); this approach is implicit also in [VY06], on which our work builds. Varacca and Yoshida provide a typing system which guarantees that the composition of confusion free event structures is confusion free. The typing is

inspired by Linear logic and Hyland-Ong strategies, and allows the authors to give an event structure semantics for (a variant of) Sangiorgi's πI-calculus.

In this paper we define composition *"operationally"*, in such a way that when restricted to strategies the machine implements strategies composition. Applied to confusion free event structures, the result is the same as the composition obtained in a more standard way. In particular, we prove the equivalence with the composition in [VY06]. We believe that the machine provides a simple and direct implementation of event structures composition.

Perspective. This work is part of a larger project, aiming at relating event structures in concurrency theory and strategies in game semantics.

The connection between event structures and strategies is especially striking if we focus on linear strategies, i.e. strategies which correspond to the multiplicative-additive structure of Linear Logic. Linear strategies (as defined in [Gir01] and then [FM05]) can be described as partial orders with a conflict relation, i.e. as a sort of event structures, which satisfy a number of conditions. In particular, they are *confusion free*. Many of the properties which make the composition work appear to depend only on confusion freeness.

Our aim on the long term is to see confusion free event structures as a generalization of strategies (i.e. as morphisms), and the composition of such event structures as strategies composition. The machine we introduce in this paper allows us a common presentation of composition. In further work [FP] we develop the game semantical structures.

2 Background

2.1 A sketch of strategies composition

In Game Semantics, the execution of a program is modelled as interaction between two players; let us call them P (Proponent) and O (Opponent). The role of a strategy is to tell the player how to respond to a counter-player move. The dialogue between the two players will produce an interaction (a play).

Figure 1 presents a simplified example of two (sequential) strategies. A specific move is played by (belongs to) only one of the players, so there are P-moves and O-moves. The active (positive) move of P are those that P plays, while its passive (negative) moves are those played by O, and to which P has to respond. In the picture, for each player strategy we distinguish the actives (positive) moves, i.e. those which belong to that player, with circles.

Let us look at the strategies (1). According to the P-player strategy, it will start with b_0, then respond with a_0 to Opponent move b_1, and with † (termination) to Opponent move b_2. Let us make it interact with the O-player strategy. The interaction goes as follows: O answer to b_0 is b_1, P answer to b_1 is a_0, O answer to a_0 is a_1, and so on.

The algorithm to calculate the interaction is simple. (i.) Start from P-player initial move, (ii.) Check counter-player answer to that move, that is, go to the corresponding opposite action, and take the following move. (iii.) Repeat step (ii.)

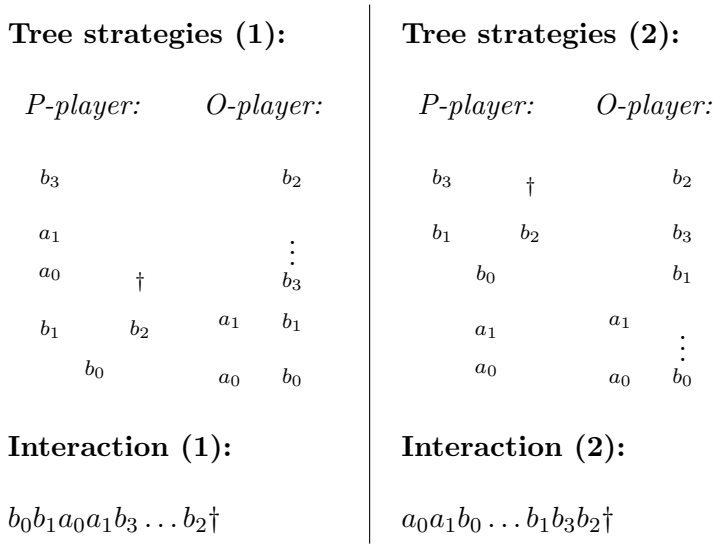


Fig. 1. Tree strategies

until terminating on †.

Figure 2 illustrates the same ideas for more parallel strategies [FM05].

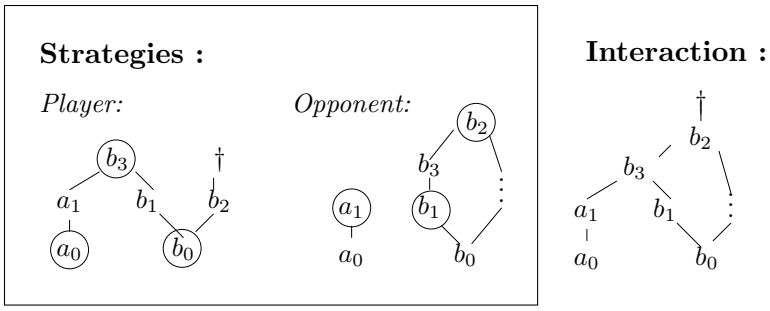


Fig. 2. Graph strategies

The strategies are here graphs. The way to make them interact is similar to the previous one, but (1.) there are several threads running in parallel, (2.) on certain moves we need to synchronize.

2.2 Event structures

Let $(\mathcal{E}, \leq)$ be a partially ordered set. Elements of $\mathcal{E}$ are called *events*; we assume that $\mathcal{E}$ is at most countable. The order relation is called *causality relation*. The *downward closure* of a subset $S \subseteq \mathcal{E}$ is defined by $\lceil S \rceil = \{e' : e' \leq e, e \in S\}$. For a singleton, we write $\lceil e \rceil$.

An **event structure**[4] is a triple $(\mathcal{E}, \leq, \smile)$ such that

- $(\mathcal{E}, \leq)$ is a partial order, as above;
- For every $e \in \mathcal{E}$, $\lceil e \rceil$ is finite.
- $\smile$ is an irreflexive and symmetric relation, called *conflict relation*, which satisfies the following:

[4] In this paper we say event structures always meaning *prime event structures*.

for every $e_1, e_2, e_3 \in \mathcal{E}$, if $e_1 \leq e_2$ and $e_1 \smile e_3$ then $e_2 \smile e_3$.

If $e_1 \leq e_2$ we say that the conflict $e_2 \smile e_3$ is *inherited* from the conflict $e_1 \smile e_3$. If a conflict is not inherited, we say that it is *immediate*, written $\smile_\mu$

Causal order and conflict are mutually exclusive.

With a slight abuse of notations, we identify an event structure $(\mathcal{E}, \leq, \smile)$ and its set $\mathcal{E}$ of events.

$[e) = \lceil e \rceil \backslash \{e\}$ is the enabling set of e; *Parents*(e) will denotes the set of immediate predecessors of e (the preconditions of that event).

A **labelled event structure** is an event structure $\mathcal{E}$ together with a labelling function $\lambda : \mathcal{E} \to L$, where L is a set of labels.

A set $S \subseteq \mathcal{E}$ is **conflict free** if it does not contain any two elements in conflict; in particular, an event structure $\mathcal{E}$ is conflict free if its conflict relation is empty. Hence, a conflict free event structure is simply a partial order. Observe that, if $e \in \mathcal{E}$, then $\lceil e \rceil$ is conflict free.

2.3 Confusion free event structures

Confusion free event structures are a class of event structures where every choice is localized to *cells*, where a cell is a set of events that are pairwise in immediate conflict, and have the same enabling set.

Definition 2.1 A **cell** c is a maximal set of events such that $e, e' \in c$ implies $e \smile_m e'$ and $[e) = [e')$.

Definition 2.2 $\mathcal{E}$ is confusion free if the following holds:

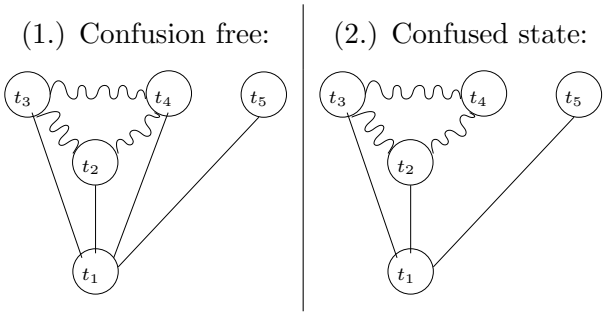
(a.) for all distinct $e, e', e'' \in \mathcal{E}$, $e \smile_\mu e'$ and $e' \smile_\mu e''$ implies $e \smile_\mu e''$

(b.) for all $e, e' \in \mathcal{E}, e \smile_\mu e'$ implies $[e) = [e')$

The notion of cell express the idea that choices are local.

2.3.1 Example.

Below, we give an example (1.) of event structure which is confusion free, and an example (2.) of an event structure which is not. Waved lines denote conflict. The event structure in (2.) is not confusion free because t_4 is in conflict with t_2, t_3, but fails to have the same parents.



3 Well-labelled event structures

In this section we introduce a notion of well-labelled event structure, where the labelling guarantees that the composition of confusion free event structures is a confusion free event structure.

Our labelling can be seen as a minimalist variant of the typing in [VY06], without the whole setting of linear types and morphisms; this because here we are only interested in the preservation of confusion freeness via composition. The labelling we use is also a straightforward generalization of the technique developed in [Gir01] to deal with additive strategies.

A key features of the labelling is that *a name identifies a cell* (rather than a single event).

3.1 Names and actions.

We assume a countable set of names N, ranged over by $\alpha, \beta, \ldots$. We are going to label a confusion free event structure with actions on these names. Let S be an index set. We define the alphabet $\mathcal{N}$ as follows:

$$\mathcal{N} = \sum_{i \in S} N_i = \{(\alpha, i) : \alpha \in N \mathit{and}\ i\ \in S\}$$

We say that $a = (\alpha, i)$ uses the name α, and write $\mathit{name}(a) = \alpha$.

A (polarized) action is given by an element $a \in \mathcal{N}$ and a polarity, which can be positive (a^+), negative (a^-), or neutral ($a^\pm$). *Polarities* correspond to one of the three main *capabilities* which a name (channel) can have: input ($-$), output ($+$), or match ($\pm$).

Remark 3.1 *Actions of opposite polarity (a^+, a^-) denote matching dual actions, such as c and $\overline{c}$ in CCS, or Player/Opponent moves in Game Semantics.*

We think of $a^\pm$ as a pair of matching actions a^+, a^- which have synchronized. A more traditional and suggestive denotation for $a^\pm$ would be τ_a.

We use the variable ϵ to vary over polarities: $\epsilon \in \{+, -, \pm\}$. When clear from the context, or not relevant, we omit the explicit indication of the polarity. The polarities $+$ and $-$ are said opposite. If a is a positive or negative action, $\overline{a}$ will denote its opposite action.

3.2 Interfaces.

We are going to type event structures with an interface. The interface provides in particular the set of names on which the event structure communicate, and their polarity.

An **interface** (A, π_A) is given by a finite set of names A, and a polarity (positive, negative, neutral) for each name. The polarization partitions A into three disjoint sets: positive, negative and neutral names.

Remark 3.2 *The positive names can be thought of as sending, the negative name as receiving, and the neutral names as private.*

The interface (A, π) generates the set of actions $\mathcal{A} = \sum_{i \in S} A_i = \{(\alpha, i) : \alpha \in A\}$. The polarization of the names extends to the actions with that name.

3.3 Well-Labelled event structures.

An event structure of interface A is a tuple $(\mathcal{E}, A, \lambda, \pi)$ where

- $\mathcal{E}$ is an event structure;
- $A = (A, \pi_A)$ is an interface;
- $\lambda : \mathcal{E} \to \{(\alpha, i) : \alpha \in A, i \in S\}$ is a labelling function;
- $\pi : \mathcal{E} \to \{+, -, \pm\}$ is the polarization induced on the actions by π_A.

If $\lambda(e) = (\alpha, i)$, we say that the event e uses the name α, and write $name(e) = \alpha$.

Remark 3.3 *If $\lambda(e) = a$, with $a = (\alpha, i)$, and $\pi_A(\alpha) = \epsilon$, then e is labelled by the action a^ϵ.*

A well-labelled event structure is guaranteed to be confusion free. We ask that: (i) all the events in the same cell use the same name (and hence also have the same polarity); (ii) two events which use the same name (and the same polarity) are in conflict.

Definition 3.4 An event structure $\mathcal{E}$ of interface A is well-labelled, written $\mathcal{E} : A$ if it satisfies the following, for all distinct $e_1, e_2 \in \mathcal{E}$.

(i) if $e_1 \smile_\mu e_2$ and $\lambda(e_1) = (\alpha, i)$, then $\lambda(e_2) = (\alpha, j)$, with $i \neq j$.

(ii) if $name(e_1) = name(e_2)$then $e_1 \smile e_2$.

(iii) $e_1 \smile_\mu e_2 \Rightarrow Parents(e_1) = Parents(e_2)$

3.4 Properties of the labelling

$\mathcal{E}$ is well-labelled iff and only if it is confusion free.

Each event $e \in \mathcal{E}$ is uniquely identified by the set of labels $\lambda\lceil e \rceil = \{\lambda e', e' \leq e\}$.

Proposition 3.5 *Given $e_1 \neq e_2 \in \mathcal{E}$, we have that $\lambda\lceil e_1 \rceil \neq \lambda\lceil e_2 \rceil$*

The conflict relation in well-labelled event structures can be inferred from the labels:

Proposition 3.6 *Let $\mathcal{E} : A$ a well-labelled event structure and let $e_1, e_2 \in \mathcal{E}$. Then the following holds:*

$$(**) \quad e_1 \smile e_2 \iff \exists e_1' \leq e_1, e_2' \leq e_2 : name(e_1') = name(e_2')$$

Since in a well-labelled event structure the labels carry all the information on the conflict relation, from now on, *we deal with the conflict implicitly*: two distinct events e_1 and e_2 are in conflict iff (**) holds.

This allows us to focus only on the partial order.

It is easy to see that

Proposition 3.7 *Given $e_1 \neq e_2 \in \mathcal{E} : A$, we have that $e_1 \smile_\mu e_2$ iff*

- *e_1, e_2 use the same name*

- $[e_1) = [e_2)$

3.4.1 Well-Labelled event structures as dag's

As seen in the previous section, given a well-labelled event structure, we can deal with the conflict implicitly; we are left to deal only with the partial order $\mathcal{E}, \leq$.

In the following, it will be convenient to identify the partial order on $\mathcal{E} : A$ with the associated dag. This in particular allows us to describe composition in terms of graph rewriting.

A *directed acyclic graph* (dag) G is an oriented graph without (oriented) cycles. We will write $c \leftarrow b$ if there is an edge from b to c. It is standard to represent a strict partial order as a dag, where we have a (non transitive) edge $a \leftarrow b$ whenever there is no c such that $a < c$ and $c < b$. Conversely (the transitive closure of) a dag is a strict partial order on the nodes.

In the following, we will identify the partial order on $\mathcal{E} : A$ with the associated dag. We take as canonical representative of $\mathcal{E}$ its *skeleton* (the minimal graph whose transitive closure is the same as that of $\mathcal{E}$).

Remark 3.8 *Observe that, by construction, the skeleton is always defined, even if $\mathcal{E}$ can have a countable number of events (in particular, a cell can have a countable number of events). In fact, for each event $e \in \mathcal{E}$, $\lceil e \rceil$ does not contain any conflict, and it is finite.*

4 Composition

We define composition *"operationally"*, in such a way that when restricted to strategies this procedure produces strategies composition.

Composition between event structures relies on two notions: synchronization and enabling (reachability). Intuitively, to compose, we synchronize (match) events which are labelled by the same action, and opposite polarity. The synchronization is possible only between events which have been *enabled*. We enable (reach) an action only if all the actions before it have been enabled (reached). We better illustrate this in Section 4.2.

4.1 Compatible interfaces

We compose two event structures when their interfaces are able to communicate.

Definition 4.1 [Compatible interfaces] Let (A, π_A) and (C, π_C) be two interfaces. The interfaces A and C are *compatible* if

for all $b \in A \cap C$, the polarity of b in A is opposite to the polarity of b in C.

If the interfaces are compatible, we define their composition $A \odot C = (A \cup C, \pi)$ where $\pi(\alpha)$ is $\pm$ if $\alpha \in A \cap C$, and otherwise as originally in A or C.

Definition 4.2 Given two compatible interfaces A, C, we say that the name α is **private** if $\alpha \in A \cap C$, **public** otherwise.

Example. If $A = \{a^-, b^+\}$ and $C = \{b^-, c^+\}$, then $A \odot C = \{a^-, c^+, b^\pm\}$. The name b is private, while the names a, c are public.

Composition is only defined on event structures which have compatible interfaces.

4.2 Conflict free composition

Let us first analyze composition in the case of conflict free event structures, i.e. when no two events are in conflict. This case is very simple and clear, but contains all the dynamics of the general case.

The key property of this case is the following

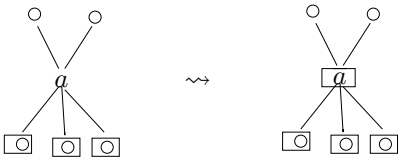
If $\mathcal{E} : A$ is conflict free, *no two events use the same name.*

Through this section, let us assume that $\mathcal{E}_1 : A$ and $\mathcal{E}_2 : C$ are conflict free and have compatible interfaces. Their composition $\mathcal{E}_1 \| \mathcal{E}_2$ is a conflict free event structure of interface $A \odot C$.

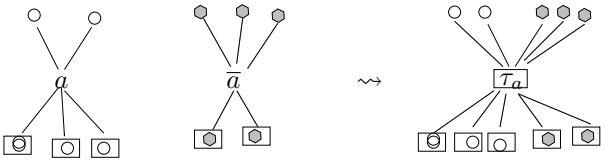
Let us describe the composition by means of a wave of tokens travelling up on $\mathcal{E}_1 \uplus \mathcal{E}_2$. When a private action is reached, to continue, it is necessary to synchronize it with an action of opposite polarity. Observe that, by construction, there is at most one such action.

Remark 4.3 *In $\mathcal{E}_1 \uplus \mathcal{E}_2$ there is only one occurrence of each polarized action. For this reason, in this section, we can identify each event with the polarized action which labels it.*

1. If a is public, and its parents have been enabled, then a is enabled. We illustrate this in the picture below, where the squares mark the enabled nodes.



2. If a is private, a^+, a^- are both present, and their parents have been enabled, then a is enabled, and the graph is transformed as follows:



end: The actions which have not been enabled are deleted (garbage collection).

The process above generates a new conflict free event structure $(E, \leq)$, where E is a set of events labelled by the actions which have been enabled; the actions have the polarity induced by the new interface $A \odot C$.

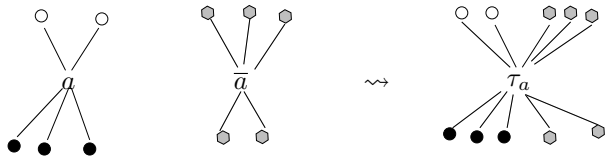
4.3 Local rewriting rules

The process described in Section 4.2 can be expressed by means of a set of local graph rewriting rules on $\mathcal{E}$, which we describe in Figure4.3.

It is straightforward to show these rules are confluent. By using this fact, one

Private a:

1. There are $a, \overline{a}$, parallel



2. Otherwise:

Fig. 3. Graph Rewriting Rules

can prove associativity for the composition.

Proposition 4.4 (Associativity) *Let $\mathcal{E}_1 : A, \mathcal{E}_2 : C, \mathcal{E}_3 : D$ be conflict free event structures. If the interfaces allow the composition, we have that*

$$(\mathcal{E}_1 \| \mathcal{E}_2) \| \mathcal{E}_3 = \mathcal{E}_1 \| (\mathcal{E}_2 \| \mathcal{E}_3)$$

4.4 Reducing composition to conflict free composition

A confusion free event structures $\mathcal{E}$ can be seen as a superposition of conflict-free event structures (which we call the slices of $\mathcal{E}$). The study of confusion free event structures can be reduced to the study of conflict free event structures. In particular, composition of confusion free event structure can be reduced to the composition of its slices.

4.4.1 Slices

A slice $\mathcal{S}$ of $\mathcal{E}$ is a downward closed, conflict free subset of $\mathcal{E}$, with the order induced by $\mathcal{E}$. To choose a (maximal) slice of $\mathcal{E}$ corresponds to the selection of a single element in each cell of $\mathcal{E}$.

4.4.2 Studying composition by slices

A key feature of the composition is that it takes place *independently* inside each single slices (Proposition 4.10) .

Several interesting properties of the composition of two event structures (such as confluence, or deadlocks) can be analyzed as properties of their slices (see 4.9).

Actually, following an approach which is well studied for proof nets and linear strategies, the process of composition itself could be reduced purely to conflict free composition:

- decompose $\mathcal{E}$ into its slices
- compose all slices pairwise

- *superpose* the composed slices

Proposition 3.6 allows us to perform the superposition, by using the same technique developed in [Gir01,FM05].

We do not give details here; however, after providing a direct description of composition in the general case, we show that the study of the composition can be reduced to the study of conflict free composition (Proposition 4.10).

4.5 Global composition

In this section, we provide a direct description of composition of well-labelled event structures, in the general case. The machine generates $\mathcal{E} = \mathcal{E}_1 \| \mathcal{E}_2$ step by step; each time we add to $\mathcal{E}$ an event v which refers to [comes from] an event (or a pair of matching events) x in $\mathcal{E}_1 \uplus \mathcal{E}_2$. We add v to $\mathcal{E}$ only if:

- the enabling set of x has already an "image" in $\mathcal{E}$;
- this image is conflict free.

Given a labelled event structure $\mathcal{E}$, and a set of events $S \subseteq \mathcal{E}$, we use the notation $\lambda S = \{\lambda(s) | s \in S\}$.

Let $\mathcal{E}_1 : A$ and $\mathcal{E}_2 : C$ be well-labelled event structures with compatible interfaces. $\mathcal{E}_1 \| \mathcal{E}_2$ is an event structure of interface $A \odot C$, obtained as follows.

Case 1. Let $e \in \mathcal{E}_i$ such that $\lambda(e) = a$ and $name(a)$ is public.
If $S \subseteq \mathcal{E}$ satisfies the following conditions:
parent condition: $\lambda S = \lambda[e)$.
conflict condition: the set S is conflict free
add to $\mathcal{E}$ an event v such that
label: $\lambda(v) = a$
edges: for all $v_i \in \lceil S \rceil$ we have $v_i \leftarrow v$

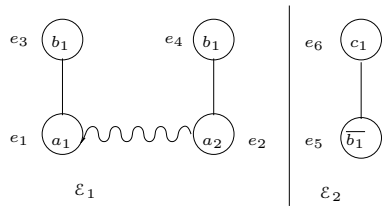
Case 2. Let $e_1 \in \mathcal{E}_1$ and $e_2 \in \mathcal{E}_2$ such that $\lambda(e_1) = \lambda(e_2) = b$ and $name(b)$ is private.
If $S = S_1 \cup S_2$ where $S_1, S_2 \subseteq \mathcal{E}$ satisfy the following conditions
parent condition: $\lambda S_1 = \lambda[e_1)$, $\lambda S_2 = \lambda[e_2)$.
conflict condition: the set S is conflict free
add to $\mathcal{E}$ an event v such that
label: $\lambda(v) = b$ (this should be thought as τ_b, since $\pi(b) = \pm$)
edges: for all $v_i \in \lceil S \rceil$ we have $v_i \leftarrow v$

The *parent condition* checks that the enabling set of $e \in \mathcal{E}_i$ has been considered, and relies on Proposition 3.6.

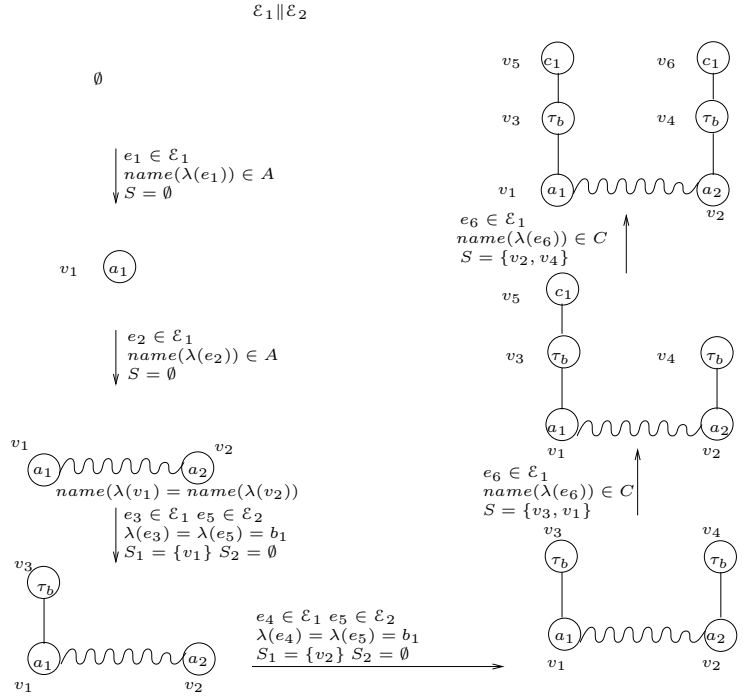
The *conflict condition* says that in $\lceil S \rceil$ there are no two events using the same names (we are using Proposition 3.7.) To understand the conflict condition, remember that events in conflict are events which are mutually exclusive. If we need a set of precondition to occur together, they must live in a conflict free event structure $\mathcal{S} \subseteq \mathcal{E}$.

The conflict condition, essentially guarantees that we are working slice by slice, i.e. independently in each slice (see Proposition 4.10)

4.5.1 Example of composition
Consider the following event structures



Here we run the machine:



4.5.2 Composition is well defined and associative
Composition of well-labelled event structures produces a well-labelled event structure. Moreover, composition is associative.

Theorem 4.5 $\mathcal{E}_1 \| \mathcal{E}_2 : A \odot C$

Proof. W.r.t. definition 3.4, conditions 1. and 2. (the properties of the labelling) hold by construction. We have to verify 3., i.e. that $u \smile_\mu v$ implies $[v) = [v)$. Let S_u, S_v the two subset of the labelled parent condition. $[u) = [v)$ if and only if $S_u = S_v$. By labelling we have that $name(\lambda(u)) = name(\lambda(v))$ public or private. We develop the public case: the other is analogous. Without loss of generality we can assume $name(\lambda(u)) \in A$. Hence there exists $e, d \in \mathcal{E}_1$ such that $\lambda S_u = \lambda[e)$ and $\lambda S_v = \lambda[e)$. We have that $e \smile_\mu d$: this holds (by 1. and 2.) and this conflict cannot be inherited because otherwise also $u \smile v$ should be inherited. Hence we have $\lambda S_u = \lambda S_v$ by confusion freeness of $\mathcal{E}_1$ and as immediate consequence of Proposition 3.6, we have $S_u = S_v$, as required. □

Remark 4.6 *As a consequence, composition of confusion free event structures is confusion free.*

Proposition 4.7 (Associativity) *Given $\mathcal{E}_1 : A, \mathcal{E}_2 : C, \mathcal{E}_3 : D$, if the interfaces allow the composition, we have that*

$$(\mathcal{E}_1 \| \mathcal{E}_2) \| \mathcal{E}_3 = \mathcal{E}_1 \| (\mathcal{E}_2 \| \mathcal{E}_3) =$$

Proof. The result follows from Proposition 4.4 and Proposition 4.10. □

4.5.3 Working by slices

Proposition 4.8 (Slices) *Let $\mathcal{E} = \mathcal{E}_1 : A \| \mathcal{E}_2 : C$. We have the following.*

- *If $\mathcal{S} \subseteq \mathcal{E}$ is a slice of $\mathcal{E}$, then there exist two slices $\mathcal{S}_1 \subseteq \mathcal{E}_1$ and $\mathcal{S}_2 \subseteq \mathcal{E}_2$ such that $\mathcal{S} = \mathcal{S}_1 : A \| \mathcal{S}_2 : C$.*
- *If $\mathcal{S}_1 \subseteq \mathcal{E}_1$ and $\mathcal{S}_2 \subseteq \mathcal{E}_2$ are slices, then $\mathcal{S} = \mathcal{S}_1 \| \mathcal{S}_2$ is a slice of $\mathcal{E}$.*

By reducing composition to composition of conflict free event structures, we can easily prove associativity.

5 Discussion

5.1 Relating with standard event structure composition

The abstract machine we have defined produces the same result as a "standard" approach to event structure composition. To do this, we choose a specific synchronization algebra, which is that defined in [VY06].

The typing defined by Varacca and Yoshida guarantees that the composition preserves confusion freeness, and allows the interpretation of a linear fragment of the π calculus.

The labelling induced by their typing is easily seen as a case of the labelling we define here, hence in particular we can apply our machine.

We have that

Proposition 5.1 *If $\mathcal{E}_1, \mathcal{E}_2$ are confusion free event structures which are typed in the sense of [VY06], they are also well-labelled in the sense defined here, and their composition $\mathcal{E}_1 \| \mathcal{E}_2$ as defined here is isomorph to the parallel composition as defined in [VY06].*

The details are given in [Pic06].

Let as briefly resume the "standard approach".

5.1.1 Parallel composition of event structures

A more standard definition for the parallel composition of event structures is that used [VY06], based on the following ingredients:

(i) fix through a synchronization algebra the events which will synchronize and those which will not. Two events synchronize if they have dual labels (for example one event has label a and the other $\overline{a}$);

(ii) build the categorical product of the event structures

(iii) discard some events, and everything above them:
   (a) discard all the events of the product which are generated from pair which are not able to synchronize because they do not have matching labels
   (b) discard all the events of the product which are generated from a single private event: these are events which are private but not "consumed".

5.2 Linear strategies with parallel composition are a sub-class of well-labelled event structures

Linear strategies as introduced in [Gir01] and extended to dag's in [FM05] are labelled dag's. The labels are taken in

$$\sum_{i \in \mathcal{P}_{fin}(\mathbb{N})} N_i = \{(\alpha, i) : \alpha \in N and\ i\ \in \mathcal{P}_{fin}(\mathbb{N})\}$$

where N are the strings of natural numbers.

The labelling satisfies a certain discipline, which in particular satisfies all the constraints in Definition 3.4.

As for composition, the machine introduced here extends the LAM machine defined in [Fag02,FM05] to implement the composition of linear strategies. The new machine has the same behaviour of the LAM when restricted to strategies. This in particular means that there is a morphism from strategies to well-labelled event structures, which preserves the parallel composition.

More precisely, strategies composition decomposes into parallel composition plus hiding, where parallel composition is the operation we have described here, and the hiding concerns the τ actions.

5.3 Bridging Game Semantics and event structures.

Our aim is to use event structure as a guide to generalize the definition of strategies. First results in this directions are presented in [FP], which builds on the machine we introduce here, and extends it to a typed setting.

On the long term, we hope to build on the body of work on event structures to extend the approach of Game Semantics to concurrency, and in particular to deal with non determinism and process calculi.

5.4 The dynamics

The machine we have presented makes it immediate what is going on when composing two labelled event structures $\mathcal{E}_1, \mathcal{E}_2$: we merge together the structure (events, order and conflicts) of $\mathcal{E}_1, \mathcal{E}_2$ to create a new event structure $\mathcal{E}$. The dynamics appears the same as that which takes place when composing strategies, λ-terms or Linear Logic proof-nets.

5.5 Event structures and proof-nets

This work meet also another line of research, which is bringing together graph strategies and proof-nets. Proof-nets are a graph representation of proofs introduced by Girard in Linear Logic [Gir87] and which are powerful tool for the analysis of normalization. In particular, they have been a fertile tool in the study of functional programming, in particular for optimization. Observe that proof-net normalization is performed via local rewriting rules.

We see event structures as a form of multiplicative-additive proof-nets, and hope to be able to apply some of the technology developed for proof-nets. For example, a key notion in proof-nets is that of correctness criterion, which states that there are no cyclic path, for a certain definition of path which is sensitive to the polarity of the nodes. The correctness criterion has a crucial role in guaranteeing that the normalization (composition) works, and in fact it guarantees that there are no deadlocks. We intend to investigate if a similar notion could be used on event structure, for an opportune typing, to guarantee that there are no deadlocks.

References

[AM99] S. Abramsky and P.-A. Mellies. Concurrent games and full completeness. In *Proceedings 15th Annual Symposium on Logic in Computer Science*, 1999.

[CF05] P.-L. Curien and C. Faggian. L-nets, strategies and proof-nets. In *CSL 05 (Computer Science Logic)*, LNCS. Springer, 2005.

[Fag02] C. Faggian. Travelling on designs: ludics dynamics. In *CSL'02 (Computer Science Logic)*, volume 2471 of *LNCS*. Springer Verlag, 2002.

[FM05] C. Faggian and F. Maurel. Ludics nets, a game model of concurrent interaction. In *Proc. of LICS'05 (Logic in Computer Science)*. IEEE Computer Society Press, 2005.

[FP] C. Faggian and M. Piccolo. Event structures and linear strategies. submitted.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir01] Jean-Yves Girard. Locus solum. *Mathematical Structures in Computer Science*, 11:301–506, 2001.

[GM04] Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. In *FOSSACS*, 2004.

[HS02] M. Hyland and A. Schalk. Games on graphs and sequentially realizable functionals. In *LICS 02*, pages 257–264. IEEE, 2002.

[Lai05] J. Laird. A game semantics of the asynchronous pi-calculus. In *Concur 05*, volume 3653 of *LNCS*, 2005.

[Mel04] P.-A. Mellies. Asynchronous games 2 : The true concurrency of innocence. In *CONCUR 04*, volume 3170 of *LNCS*. Springer Verlag, 2004.

[MW05] G. McCusker and M. Wall. Categorical and game semantics for scir. In *Galop 2005*, pages 157–178, 2005.

[NPW81] M. Nielsen, G. Plotkin, and G. Winskel. Event structures and domains 1. *Theoretical Computer Science*, 13:85–108, 1981.

[NYB01] K. Honda N. Yoshida and M. Berger. Sequentiality and the pi-calculus. In *Proc. of TLCA 2001, the 5th International Conference on Typed Lambda Calculi and Applications*, LNCS. Springer, 2001. Extended abstract.

[Pic06] M. Piccolo. Event structures and strategies. Master's thesis, Dip. Matematica Pura e Applicata, Universitá di Padova, 2006.

[SPP05] A. Schalk and J.J. Palacios-Perez. Concrete data structures as games. In *CTCS 04*, volume 122 of *Electr. Notes Theor. Comput. Sci.*, 2005.

[VY06] D. Varacca and N. Yoshida. Typed event structures and the pi-calculus. In *MFPS*, 2006.