PARALLEL PRODUCT OF EVENT STRUCTURES

Ilaria Castellani

INRIA, Sophia-Antipolis, 06560 Valbonne, France

Guo-Qiang Zhang

Department of Computer Science

The University of Georgia

Athens, Georgia 30602, U.S.A

1 Introduction

Event structures were introduced by Nielsen, Plotkin and Winskel in [NPW81] as a model for computational processes. These were essentially sets of events with relations expressing causal dependencies and conflicts between them. These structures, also called *prime event structures*, were shown to be connected both with a class of acyclic Petri nets (called by the authors *occurrence nets*) and with a class of domains (finitary prime algebraic coherent domains). Though a very pleasant model of computation, prime event structures can lead to tedious definitions for more sophisticated constructions such as parallel product or exponentiation. In subsequent papers ([Win82, Win87]), Winskel introduced a more general class of event structures called *stable event structures*, for which such constructions are defined categorically. Although stable event structures themselves are slightly more complicated than prime event structures, constructions on them can be defined in an elegant way. In general, category theory has proven very useful for computer science. When constructions are formulated in categorical terms, not only can the required universal properties provide some guideline in the definitions, the associated morphisms may contain suggestions of a computational interest as well.

Moving to more general structures, however, can sometimes shield the intuitions about the model. In the case of stable event structures, the causality relation on events is not explicitly given, but is derived from an *enabling* relation on events. This makes such event structures a little hard to visualize, and one may wonder whether a more manageable notion could be devised. Moreover, unlike prime event structures, it is not clear if stable event structures correspond to a class of Petri nets, in the sense of [NPW81].

Flow event structures were proposed in [BC88] as an intermediate between prime event structures and stable event structures, suitable both for graphical representation and for an easy definition of process operators. Among these operators the critical one is the parallel product. A "parallel product" can be defined easily on flow event structures. The problem is that it may not be a categorical one. This is unsettling, because if a "product" is not categorical, we are left to wonder what it is. The aim of this note is to show that, for flow event structures satisfying a particular constraint – which is preserved by usual process operators – the product is indeed categorical. We mention in passing that flow event structures do have a direct connection with a class of Petri nets, as shown by Boudol in [Bou90]. These nets – called *flow nets* – are strictly more general than occurrence nets: they may have cycles in the flow relation although they are still "semantically" acyclic.

Like stable event structures, flow event structures determine the same class of domains as prime event structures (*cf* again [Bou90]). Thus for any construction on flow (or stable) event structures one may obtain a corresponding construction on prime event structures, by passing through their domains of configurations. In fact, direct definitions of parallel product on prime event structures – although rather complicated – have been given explicitly by Goltz and Loogen in [GL91] and by Degano, De Nicola and Montanari in [DDNM88]. More recently a simpler definition was proposed by Vaandrager in [Vaa89].

2 Flow event structures and product

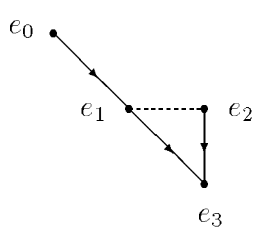
Flow event structures are a direct generalisation of prime event structures (we refer to [NPW81] for the definition of prime event structures) where the conflict relation is not inherited and the partial ordering of causality is replaced by a local *flow* relation on events. Intuitively, the flow relation represents an immediate causality between two events. However, since events may occur in different ways (as in stable event structures) any causal dependency is merely "possible", and makes sense only within computations. A simple way of understanding the flow relation is by analogy with Petri nets: a flow between two events in an event structure corresponds to the presence of a condition between the events in a net. This point is illustrated by the example after the following basic definition.

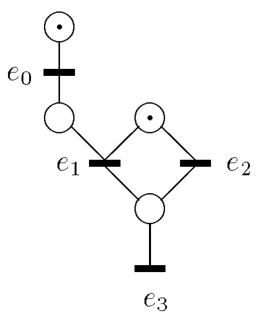
Definition 2.1 *(Flow event structures) A flow event structure is a triple $S = (E, \# , \prec)$ where:*

- E *is a denumerable set of* events,
- $\# \subseteq (E \times E)$ *is a symmetric* conflict *relation, and*
- $\prec \subseteq (E \times E)$ *is an irreflexive* flow *relation.*

It should be clear that any prime event structure $S = (E, \# , \leq)$ is a flow event structure (with $\prec$ given by the strict ordering $<$). Note that the flow relation is not required to be transitive, nor its closure $\prec^*$ to be acyclic. Also, the conflict relation is not assumed to be irreflexive. This means that there may be *self-conflicting* or *inconsistent* events, and we will see that these are essential for defining some constructions on flow event structures – namely restriction and specialised parallel products like CCS parallel composition. Inconsistent events also play a crucial role in Boudol's constructions between flow event structures and flow nets [Bou90].

The following is an example of a flow event structure, together with the "corresponding" Petri net. In drawings, we represent $e \prec e'$ by a directed arc $e \longrightarrow e'$ and $\#$ by a dotted line. Self-conflicts will be represented by dotted circles around events.





This example (which typically arises when modelling CCS communication) exhibits both a confluence after conflict, and a case where the flow $\prec$ is essentially *not* transitive: the events e_0 and e_3 are indeed causally related if e_1 occurs, but they are independent if e_2 occurs. In other words, e_0 and e_3 are in a different relation depending on the computation where they are considered. For more examples of flow event structures we refer the reader to [BC88].

We shall now formalise this notion of computation or *configuration* for flow event structures. A configuration is a set of events having occurred at some stage of evolution of a process. Since flow event structures are rather general, the definition of configuration is slightly more elaborated than for prime event structures. Let Con_S be the set of conflict-free (consistent) sets of events: $X \in \mathsf{Con}_S$ iff $\forall e, e' \in X, \neg(e \,\#\, e')$. Obviously, an event e is inconsistent ($e \,\#\, e$) if and only if $\{e\} \notin \mathsf{Con}_S$. For a subset X of E, let $\prec_X$ be the restriction of the flow relation to X and $\leq_X =_{def} {\prec_X}^*$ be the preordering generated by $\prec_X$. For simplicity, we consider here only finite configurations – see [BC88] for a more general treatment of configurations.

Definition 2.2 *(Configurations) Let $S = (E, \,\#\, , \prec)$ be a flow event structure. A (finite) configuration of S is a finite subset X of E such that:*

1. *X is conflict-free: $X \in \mathsf{Con}_S$,*

2. *X is left-closed up to conflicts:*
   $e' \prec e \in X \;\&\; e' \notin X \implies \exists\, e'' \in X \,.\, e' \,\#\, e'' \prec e$, *and*

3. *X has no causality cycles: the relation $\leq_X$ is an ordering.*

The first two conditions are essentially the same as for prime event structures: condition 2) is adapted to account for the more general, non-hereditary, conflict relation. It states that any event appears in a configuration with a “complete” set of causes. Condition 3) ensures that any event in a configuration is actually reachable at some stage of computation. Note that an inconsistent event cannot appear in a configuration. So for example the structure

$$\begin{tikzcd} e \arrow[d] \\ e' \end{tikzcd}$$

has only the trivial configuration $\emptyset$. The set of (finite) configurations of a flow event structure S will be denoted by $\mathcal{F}(S)$.

We have seen that prime event structures are a subclass of flow event structures. In turn, any flow event structure $S = (E, \#, \prec)$ may be described as a stable event structure $G_S = (E, \#', \vdash_S)$ such that $\mathcal{F}(G_S) = \mathcal{F}(S)$. This is explained in [BC88, Bou90]. We recall here briefly the definition of G_S.

The enabling relation $\vdash_S \subseteq \mathcal{P}(E) \times E$ is defined as follows. Say that "e is a condition for e'" when $e \prec e'$. Then $F \vdash_S e$ holds whenever F is a maximal set of non-conflicting conditions for e, that is:

$$F \vdash_S e \quad \Longleftrightarrow_{\text{def}} \begin{cases} e' \in F \;\Rightarrow\; e' \prec e \\ F \cup \{e\} \text{ is consistent : } \forall e', e'' \in F \cup \{e\} : \neg(e' \,\#\, e'') \\ F \text{ is closed under non-conflicting conditions for } e : \\ e' \prec e \;\&\; e' \notin F \;\Rightarrow\; \exists\, e'' \in F \text{ s.t. } e' \,\#\, e'' \prec e \end{cases}$$

Note that since $F \cup \{e\}$ must be consistent w.r.t. $\#$, an event e which is inconsistent in S will have no enabling set in G_S. Then the conflict $\#'$ is just the irreflexive restriction of $\#$, that is $\#' = \# - Id$, where Id is the identity relation on events. It is easy to see that the structure $G_S = (E, \#', \vdash_S)$ obtained in this way is a stable event structure in the sense of Winskel, with $\vdash_S$ being the minimal enabling.

On the other hand a stable event structure cannot always be represented as a flow event structure (cf again [Bou90]). Hence flow event structures are strictly included between prime and stable event structures.

We shall now define the parallel product on flow event structures. We use Winskel's notation $*$ for undefined values of partial functions and write $e \mathrel{\mathsf{W}} e'$ for the reflexive closure of $\#$, that is $e \mathrel{\mathsf{W}} e' \Longleftrightarrow_{def} (e \,\#\, e' \text{ or } e = e')$. Here and in what follows, an assertion $f(e)\ R\ f(e')$ – where f is a partial function on events and $R \in \{\prec, \#, =\}$ – will imply that both $f(e)$ and $f(e')$ are defined. We shall mostly write fe for $f(e)$.

Definition 2.3 *(Parallel product) Let* $S_0 = (E_0, \#_0, \prec_0)$ *and* $S_1 = (E_1, \#_1, \prec_1)$ *be flow event structures. Their parallel product* $(S_0 \times S_1)$ *is the event structure* $S = (E, \#, \prec)$ *defined by:*

i) $E =^{\mathrm{def}} (E_0 \times_* E_1) =^{\mathrm{def}} \{(e_0, *) \mid e_0 \in E_0\} \cup \{(*, e_1) \mid e_1 \in E_1\} \cup$
$\{(e_0, e_1) \mid e_0 \in E_0\ \&\ e_1 \in E_1\}$,

ii) $e \mathrel{W} e' \iff (\pi_0 e \mathrel{W} \pi_0 e')$ or $(\pi_1 e \mathrel{W} \pi_1 e')$,

iii) $e \# e \iff (\pi_0 e \# \pi_0 e)$ or $(\pi_1 e \# \pi_1 e)$, *and*

iv) $e \prec e' \iff (\pi_0 e \prec \pi_0 e')$ or $(\pi_1 e \prec \pi_1 e')$.

where the projections $\pi_i : E \longrightarrow_* E_i$ *are given by* $\pi_i(x_0, x_1) = x_i$, *for* $i = 0, 1$.

Condition $iii)$ explicitly deal with self-conflicts. It states that the product inherits self-conflicts from its components, and never introduces any new ones.

We now need the notion of a morphism to define a category of flow event structures.

Definition 2.4 *(Morphisms) Let* $S_0 = (E_0, \#_0, \prec_0)$ *and* $S_1 = (E_1, \#_1, \prec_1)$ *be flow event structures. A morphism from* S_0 *to* S_1 *is a partial function* $f : E_0 \longrightarrow_* E_1$ *satisfying:*

i) $fe = fe' \implies e \mathrel{W} e'$,

ii) $fe \# fe' \implies e \# e'$,

iii) $fe \prec fe' \implies e \prec e'$, *and*

iv) $X \in \mathcal{F}(S_0) \implies f(X) \in \mathcal{F}(S_1)$.

The intuition for morphisms is relatively well-understood, and it is similar to that of Winskel for stable event structures. A morphism $f : E_0 \longrightarrow_* E_1$ on flow event structures expresses the synchronisation of an event $e \in E_0$ with the event $f(e)$. Condition i) says that two distinct events e, e' can synchronise with a common event $f(e)$ only when they are in conflict, which makes sure that this kind of synchronisation can never happen in a computation. Condition ii) says that the partial function f preserves consistency: a morphism does not create new conflicts. In particular, it ensures that morphisms do not create new self-conflicts: if $fe \# fe$, this is always because $e \# e$. Condition iii) says a morphism preserves causality, as well.

It needs an explanation why condition iv) is explicitly required here, because for stable event structures, this property follows from similar conditions to $i) - -iii)$. Suppose X is a configuration of S_0. Clearly, $f(X)$ remains conflict-free. However, to prove that $f(X)$ is left-closed up to conflicts, we need to show

$$\eta \prec f(e) \in f(X) \ \& \ \eta \notin f(X) \implies \exists\, f(e'') \in f(X) \, . \, \eta \# f(e'') \prec f(e).$$

Unfortunately, the given condition for $f(X)$ does not in general translate into a similar condition for X for us to apply the corresponding property of X, because η may *not* be in the image $f(E_0)$. We do get the above iv), though, if we require that f is *down-onto*:

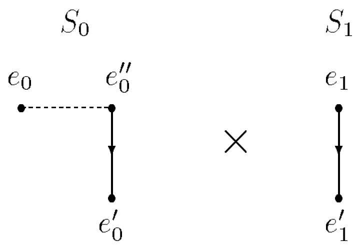
$$\eta \prec f(e) \implies \exists e' \in E_0 \, . \, \eta = f(e'),$$

which is weaker than iv). Since this is not the main concern of this paper, we do not speculate further on it here.

It is now easy to establish the following:

Fact 2.5 *Flow event structures with morphisms form a category with the composition of partial functions as composition and the identity functions as identity morphisms.*

It is natural to expect that the product construction $S_0 \times S_1$ introduced earlier is indeed the product in the categorical sense. Unfortunately, this is not the case, as shown by the following counterexample.



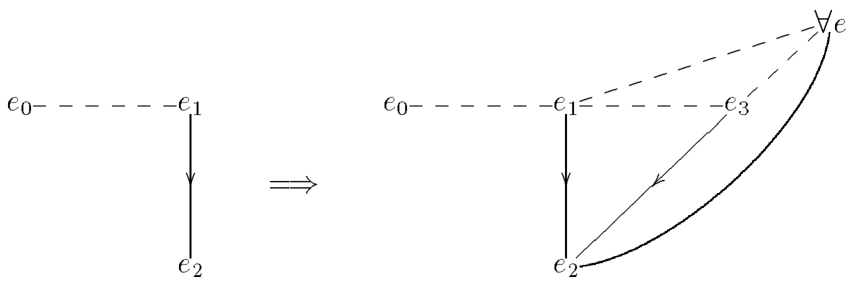
We let the reader verify that in $(S_0 \times S_1)$ the set $\{(e_0, e_1)\}$ is a complete set of causes for (e_0', e_1') and that $X = \{(e_0, e_1), (e_0', e_1')\}$ is a configuration. However $\pi_0(X) = \{e_0, e_0'\}$ is not a configuration of S_0 (and thus the projection π_0 is not a morphism). Indeed, X should not be allowed as a configuration here, since in S_0 the event e_0' cannot occur unless e_0'' has occurred, and thus in $(S_0 \times S_1)$ the event (e_0', e_1') should not occur unless some event involving e_0'' has occurred. (One might try to modify the notion of configurations to accommodate this situation, but that will unlikely lead us anywhere because the definition of configurations is well justified.)

The problem arises with the particular form of the structure on the left-hand side: such a structure will be called a *triangle* in the rest of the paper. Formally, a triangle is a structure with three distinct events e_0, e_1, e_2 such that $e_0 \# e_1 \prec e_2$ and e_0 is not related to e_2. We will show that in structures generated by usual process constructors such triangles never occur in isolation, but are always "saturated " by other events. These additional events will precisely prevent sets like X in the example above to be admitted as configurations.

For two distinct events e, e' we write $e \sim e'$ for ($e \# e'$ or $e \prec e'$ or $e' \prec e$). In drawings we shall represent $\sim$ by an undirected arc. The structures we shall consider are those satisfying the following structural property:

Axiom Δ: $e_0 \# e_1 \prec e_2 \;\&\; e_0 \not\sim e_2 \;\Rightarrow\; \exists\, e_3.\, (e_1 \# e_3 \prec e_2) \;\&\; (\forall e \# e_3 :\; e_1 \vee\!\!\!\vee\, e \sim e_2)$.

In picture:



This axiom says that if an event e_0 is in conflict with event e_1, which is a cause for another event, e_2, then in the case that e_0 is not related to e_2 by either conflict or causal dependency, there must be some event e_3 in conflict with e_1, while at the same time being a cause for e_2. Moreover, any other event in conflict with e_3 must be in conflict with e_1 and, at the same time, be causally related to e_2. The very *rough* intuition is that when a problematic situation such as the one pictured on the left hand side of the previous diagram occurs, then there is a pivotal event, e_3, such that any event in conflict with it must provide the "missing links" for the problematic situation.

We show that for structures satisfying Δ our definition yields the desired product construction.

Proposition 2.6 *Let S_0, S_1 be flow event structures satisfying axiom Δ. Then $(S_0 \times S_1)$ is the categorical product of S_0 and S_1.*

Proof: Notation: events of S will be denoted by e, e' etc. and events of S_i by e_i, e_i'. Also, we shall write fe for $f(e)$, and $fe\downarrow$ to mean that fe is defined.

1) We first show that the projections π_i are morphisms. Conditions $i), ii)$ and $iii)$ are obviously satisfied. Thus we only have to prove that the π_i's preserve configurations. Consider for example $\pi_0 : E \longrightarrow_* E_0$. Let $X \in \mathcal{F}(S)$; we want to show that $\pi_0(X) \in \mathcal{F}(S_0)$.

It is obvious that $\pi_0(X)$ is conflict-free, since $\pi_0 e \# \pi_0 e'$ would imply $e \# e'$ in X. We show that $\pi_0(X)$ is consistently left-closed. Let $e \in X$, $e_0 \prec \pi_0 e$ & $e_0 \notin \pi_0(X)$. Then $(e_0, *) \prec$

e and$(e_0, *) \notin X$. Since X is a configuration, there exists $e' \in X$ s.t. $(e_0, *) \;\#\; e' \prec e$. By *ii)* it must be the case that $\pi_0 e' \downarrow$ and $\pi_0 e' \mathrel{\mathrm{W}} e_0$. Now $\pi_0 e' \neq e_0$ because $e_0 \notin \pi_0(X)$, thus $\pi_0 e' \;\#\; e_0$.

Suppose $\pi_0 e' \sim \pi_0 e$. We have $\pi_0 e' \prec \pi_0 e$, since otherwise X would contain a conflict (if $\pi_0 e' \;\#\; \pi_0 e$) or a loop (if $\pi_0 e' \succ \pi_0 e$, recall that $e' \prec e$). Thus we have $e_0 \;\#\; \pi_0 e' \prec \pi_0 e$.

Otherwise, if $\pi_0 e' \not\sim \pi_0 e$, we have a triangle $\pi_0 e' \;\#\; e_0 \prec \pi_0 e$ and we can use axiom Δ to deduce the following:

$$\exists e_0'.\ (e_0 \;\#\; e_0' \prec \pi_0 e)\ \&\ (\forall e_0'' \;\#\; e_0' :\ e_0 \mathrel{\mathrm{W}} e_0'' \sim \pi_0 e).$$

Then $(e_0', *) \prec e \in X$. Now if $e_0' \in \pi_0(X)$ we have finished, since $e_0 \;\#\; e_0' \prec \pi_0 e$. Otherwise, if $e_0' \notin \pi_0(X)$, we have $(e_0', *) \notin X$, hence $\exists e'' \in X$ s.t. $(e_0', *) \;\#\; e'' \prec e$. Whence $e_0' \mathrel{\mathrm{W}} \pi_0 e''$ and thus $e_0' \;\#\; \pi_0 e''$. By axiom Δ again, we have $\pi_0 e'' \sim \pi_0 e$ and $\pi_0 e'' \mathrel{\mathrm{W}} e_0$. Clearly $\pi_0 e'' \neq e_0$, since $e_0 \notin \pi_0(X)$. Also, from $e, e'' \in X$ and $e'' \prec e$, it follows that $\pi_0 e'' \prec \pi_0 e$.

It is clear that there are no loops in $\pi_0(X)$, since these would be inherited in X. We have thus proved that projections are morphisms.

2) Second, we must show that $(S_0 \times S_1)$ is canonical, i.e. that for any diagram:

$$\begin{tikzcd}
 & S_0 \times S_1 \arrow[dl, "\pi_0"'] \arrow[dr, "\pi_1"] & \\
S_0 & & S_1 \\
 & \bar{S} \arrow[ul, "\alpha_0"] \arrow[ur, "\alpha_1"'] \arrow[uu, dashed, "\theta"] &
\end{tikzcd}$$

there exists a unique $\theta : \bar{E} \longrightarrow_* (E_0 \times_* E_1)$ that makes the diagram commute. The only possible candidate is (still using e, e' for events of $\bar{E}$) $\theta :\ e \longmapsto_* (\alpha_0 e, \alpha_1 e)$. We show that θ is a morphism. We have, for any $e, e' \in \bar{E}$:

$$\begin{aligned}\theta e \prec \theta e' &\Longleftrightarrow_{def} (\alpha_0 e, \alpha_1 e) \prec (\alpha_0 e', \alpha_1 e') \\ &\Longleftrightarrow (\alpha_0 e \prec \alpha_0 e') \text{ or } (\alpha_1 e \prec \alpha_1 e') \quad \text{(by definition of product)} \\ &\Longrightarrow e \prec e' \quad \text{(because both } \alpha_i\text{'s are morphisms)}\end{aligned}$$

Similarly, for any $e, e' \in \bar{E}$:

$$\begin{aligned}\theta e \smile \theta e' &\Longleftrightarrow_{def} (\alpha_0 e, \alpha_1 e) \smile (\alpha_0 e', \alpha_1 e') \\ &\Longleftrightarrow (\alpha_0 e \smile \alpha_0 e') \text{ and } (\alpha_1 e \smile \alpha_1 e') \\ &\Longrightarrow e \smile e' \quad \text{(because the } \alpha_i\text{'s are morphisms)}\end{aligned}$$

Suppose now $\theta e \;\#\; \theta e'$. We have:

$$\theta e \;\#\; \theta e' \Longleftrightarrow_{def} (\alpha_0 e, \alpha_1 e) \;\#\; (\alpha_0 e', \alpha_1 e')$$

$$\Longleftrightarrow \begin{cases} \theta e \neq \theta e' \;\&\; (\alpha_0 e \smile \alpha_0 e' \text{ or } \alpha_1 e \smile \alpha_1 e') \\ \theta e = \theta e' \;\&\; (\alpha_0 e \;\#\; \alpha_0 e' \text{ or } \alpha_1 e \;\#\; \alpha_1 e') \end{cases} \quad \text{(by product definition)}$$

Now, suppose $\theta e \neq \theta e'$: then we cannot have both $(\alpha_0 e = \alpha_0 e')$ and $(\alpha_1 e = \alpha_1 e')$. Thus it must be $(\alpha_0 e \;\#\; \alpha_0 e')$ or $(\alpha_1 e \;\#\; \alpha_1 e')$. This implies $e \;\#\; e'$ since both α_i's are morphisms. Similarly, the case where $\theta e = \theta e'$ immediately implies $e \;\#\; e'$.

It remains to check that θ preserves configurations. Let $X \in \mathcal{F}(\bar{S})$. We want to show that $\theta(X) =_{def} \{(\alpha_0 e, \alpha_1 e) \mid e \in X\} \in \mathcal{F}(S_0 \times S_1)$. Again, it is obvious that $\theta(X)$ is conflict-free.

We show that $\theta(X)$ is consistently left-closed. Let $k = (k_0, k_1) \prec (\alpha_0 e, \alpha_1 e)$ for some $e \in X$ and $k \notin \theta(X)$. By definition of product we have $k_0 \prec \alpha_0 e$ or $k_1 \prec \alpha_1 e$. Assume $k_0 \prec \alpha_0 e$.

If $\exists e' \in X$ s.t. $\alpha_0 e' = k_0$, then $(\alpha_0 e', \alpha_1 e') \prec (\alpha_0 e, \alpha_1 e)$ by definition of $\prec$ in the product. Moreover $(\alpha_0 e', \alpha_1 e') \in \theta(X)$ and $(\alpha_0 e', \alpha_1 e') \;\#\; k = (k_0, k_1)$ because $\pi_0(\alpha_0 e', \alpha_1 e') = \alpha_0 e' = k_0$ and $(\alpha_0 e', \alpha_1 e') \neq k$. Otherwise for any $e' \in X$ we have $k_0 \neq \alpha_0 e'$. In this case, since $k_0 \prec \alpha_0 e \in \alpha_0(X) \in \mathcal{F}(E_0)$ and $k_0 \notin \alpha_0(X)$, there must be $e'' \in X$ s.t.

$\alpha_0 e'' \prec \alpha_0 e$ & k_0 # $\alpha_0 e''$. Hence $(\alpha_0 e'', \alpha_1 e'') \prec (\alpha_0 e, \alpha_1 e)$, and k # $(\alpha_0 e'', \alpha_1 e'')$ because k_0 # $\alpha_0 e''$.

Finally it is clear that there are no loops in $\theta(X)$, since these would be reflected in X. □

We show now that the parallel product preserves axiom Δ. The idea is that for any triangle e_0 # $e_1 \prec e_2$ introduced by the product, the *source* e_1 of the triangle is a compound event, which inherits its causality relation to e_2 from some atomic component e_3. This event e_3 is precisely the one which is required by axiom Δ.

Proposition 2.7 *Let S_0, S_1 be flow event structures satisfying axiom Δ. Then the structure $(S_0 \times S_1)$ satisfies Δ.*

Proof. Suppose there is a triangle e_0 # $e_1 \prec e_2$, $e_0 \not\prec e_2$ in $(S_0 \times S_1)$. We want to prove that axiom Δ is satisfied. Since $e_1 \prec e_2$ it must be $(\pi_0 e_1 \prec \pi_0 e_2)$ or $(\pi_1 e_1 \prec \pi_1 e_2)$. Assume $\pi_0 e_1 \prec \pi_0 e_2$. Since $e_0 \not\prec e_2$ we have $\pi_0 e_0 \not\prec \pi_0 e_2$ and thus $\pi_0 e_0 \neq \pi_0 e_1$. There are then three cases left for e_0 # e_1:

1. $\pi_1 e_0 = \pi_1 e_1$,

2. $\pi_0 e_0$ # $\pi_0 e_1$,

3. $\pi_1 e_0$ # $\pi_1 e_1$.

Case 1: $\pi_1 e_0 = \pi_1 e_1$. We want to show that the triangle: e_0 # $e_1 \prec e_2$ satisfies axiom Δ. Take $(\pi_0 e_1, *)$ as a candidate for e_3 in the axiom. Certainly $(\pi_0 e_1, *) \prec e_2$ and $(\pi_0 e_1, *)$ # e_1. Thus either $(\pi_0 e_1, *)$ fulfils axiom Δ or $\exists e$ # $(\pi_0 e_1, *)$ for which $(e \not\prec e_2)$ or $\neg(e \,\mathbb{W}\, e_1)$. Now it cannot be $\neg(e \,\mathbb{W}\, e_1)$ since e # $(\pi_0 e_1, *)$ implies $\pi_0 e \,\mathbb{W}\, \pi_0 e_1$. Then it must be $e \not\prec e_2$, which implies $\pi_0 e \not\prec \pi_0 e_2$. We have now a triangle $\pi_0 e$ # $\pi_0 e_1 \prec \pi_0 e_2$ in S_0. By axiom Δ , there exists $e_3 \in E_0$ s.t.

$$(e_3 \prec \pi_0 e_2 \ \& \ e_3 \ \# \ \pi_0 e_1) \ \& \ (\forall s \ \# \ e_3 : \ s \sim \pi_0 e_2 \ \& \ s \,\mathbb{W}\, \pi_0 e_1).$$

Then $(e_3, *) \in (E_0 \times_* E_1)$ is such that

$$((e_3, *) \prec e_2 \ \& \ (e_3, *) \ \# \ e_1) \ \& \ \forall r \ \# \ (e_3, *)$$

because

- if $\pi_0 r = e_3$ then $r \prec e_2$ & $r \,\#\, e_1$;

- if $\pi_0 r \,\#\, e_3$ then $r \sim e_2$ & $r \,\#\, e_1$.

Case 2: $\pi_0 e_0 \,\#\, \pi_0 e_1$. Again we want to prove Δ for the triangle $e_0 \,\#\, e_1 \prec e_2$. This is straightforward. We have a corresponding triangle in S_0 : $\pi_0 e_0 \,\#\, \pi_0 e_1 \prec \pi_0 e_2$. By axiom Δ there exists $e_3 \in E_0 . (e_3 \prec \pi_0 e_2 \;\&\; e_3 \,\#\, \pi_0 e_1)$ and $(\forall s \,\#\, e_3 : s \sim \pi_0 e_2 \;\&\; s \,\mathbb{W}\, \pi_0 e_1)$. Now if we take $(e_3, *)$ in$(E_0 \times_* E_1)$, this is such that $((e_3, *) \prec e_2 \;\&\; (e_3, *) \,\#\, e_1) \;\&\; \forall e \,\#\, (e_3, *)$:

- if $\pi_0 e = e_3$ then $e \prec e_2$ & $e \,\#\, e_1$;

- if $\pi_0 e \,\#\, e_3$ then $e \sim e_2$ & $e \,\#\, e_1$.

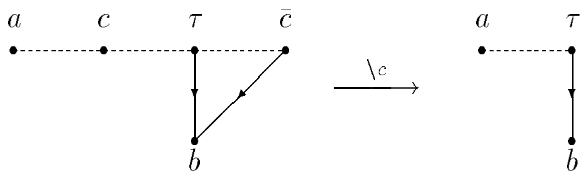
Case 3: $\pi_1 e_0 \,\#\, \pi_1 e_1$. Take again $(\pi_0 e_1, *)$ as a candidate for e_3 in Δ. We have now $(\pi_0 e_1, *) \prec e_2$ and $(\pi_0 e_1, *) \,\#\, e_1$. Now either $(\pi_0 e_1, *)$ fulfils axiom Δ, or $\exists e \,\#\, (\pi_0 e_1, *)$ for which $(e \not\sim e_2)$ or $\neg(e \,\mathbb{W}\, e_1)$. From $e \,\#\, (\pi_0 e_1, *)$ it follows that $\pi_0 e \,\mathbb{W}\, \pi_0 e_1$ and thus $e \,\#\, e_1$. Then it must be $e \not\sim e_2$, whence $\pi_0 e \not\sim \pi_0 e_2$. Again we have a triangle $\pi_0 e \,\#\, \pi_0 e_1 \prec \pi_0 e_2$ in S_0, and by axiom Δ there exists $e_3 \in E_0$ s.t. $(e_3 \prec \pi_0 e_2 \;\&\; e_3 \,\#\, \pi_0 e_1) \;\&\; (\forall s \,\#\, e_3 . s \sim \pi_0 e_2 \;\&\; s \,\mathbb{W}\, \pi_0 e_1)$. Then $(e_3, *) \in (E_0 \times_* E_1)$ is such that $((e_3, *) \prec e_2 \;\&\; (e_3, *) \,\#\, e_1) \;\&\; \forall r \,\#\, (e_3, *)$:

- if $\pi_0 r = e_3$ then $r \prec e_2$ & $r \,\#\, e_1$;

- if $\pi_0 r \,\#\, e_3$ then $r \sim e_2$ & $r \,\#\, e_1$.

□

Another important operation on flow event structures is *restriction*. The standard construction for restriction, as it features in CCS and in Winskel treatment of event structures, operates by removing all events of a given set. In fact, this construction may affect quite drastically the structure of a process, and, not surprisingly, it does not preserve property Δ. The following is an example of a triangle introduced by usual restriction (one may regard this structure as representing the CCS term $((a + c) \mid \bar{c}b) \backslash c$).

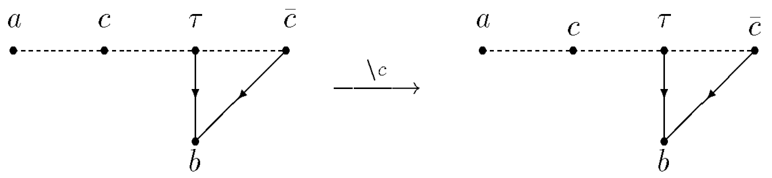


The definition of restriction proposed in [BC88] for flow event structures is actually a different one, which makes use of self-conflicting – or inconsistent – events. More precisely, restriction of a flow event structure to a set of events E' is modelled by rendering inconsistent all events not belonging to E'.

Definition 2.8 *(Restriction) Let $S = (E, \#, \prec)$ be a flow event structure and $E' \subseteq E$. The restriction of S to E' is the structure $S \upharpoonright E' = (E, \#', \prec)$ where:*

$$e \, \#' e' \iff (e \; \# \; e') \text{ or } (e = e' \;\&\; e \notin E')$$

It is easy to see that this operator of restriction preserves Δ, since it preserves events as well as their relations with *other* events. Applying restriction to the previous example gives now:



where the triangle is still saturated (like before the restriction) as required by axiom Δ.

3 Interpreting constructs with flow event structures

Flow event structures are well-suited to model languages like CCS. Moreover, all CCS operators turn out to preserve property Δ, thus we are sure to obtain the categorical product construction. A complete definition of CCS operators on flow event structures may be found in [BC88]. Here we shall just give a reformulation of CCS *parallel composition* as a restricted parallel product.

Languages like CCS are parameterized on a set of actions L. To model processes in these languages one uses event structures *labelled* on this set of actions. The synchronization discipline of the language – which actions may combine together into a synchronization action – is expressed by a *synchronization algebra* on the labels [Win82].

We use here a variant of Winskel's definition of synchronisation algebra, which seems more convenient for our purposes. We define a synchronization algebra on a set of labels L to be of the form $(L, \bullet, \omega)$ with just one special element $\omega \in L$. Here $\bullet$ is a commutative and associative operation on L, used to yield labels for pairs of events, namely: $l(x, y) =_{def} l(x) \bullet l(y)$. The role of ω is twofold:

- The label ω is used for pairs of events (x, y) which represent forbidden synchronisations. Such events must not occur in the parallel composition, and we express this fact by the axiom: $l(x, y) = \omega \implies (x, y)$ is inconsistent.

- Also, ω is used to label the component $*$ of an *asynchrony pair* – where one of the components is $*$ – i.e. the labelling function l is extended by the convention $l(*) = \omega$. We recall that $*$ is not a real event, but just a notational device (and thus will never occur in the set of events E of an event structure). Then an asynchrony pair of the form $(*, e)$ or $(e, *)$ is allowed in the product if and only if $l(e) \bullet \omega \neq \omega$.

This motivates the following definition:

Definition 3.1 *An* L-labelled flow event structure *is a structure* $S = (E, \#, \prec, l)$ *where* $(E, \#, \prec)$ *is a flow event structure and* $l : E \longrightarrow L$ *is a labelling function over a set* L *of labels such that* $\omega \in L$ *and* $l(e) = \omega \implies e \# e$.

Note that while all ω-labelled events are inconsistent, there may still be inconsistent events whose label is different from ω.

The operations of product and restriction are extended to labelled structures in the obvious way. In the product $(S_0 \times S_1)$ the label of an event e is defined to be $l_0(\pi_0(e)) \bullet l_1(\pi_1(e))$. In the restriction $S \upharpoonright L'$, where $L' \subseteq L$, all events carrying a label in $(L - L')$ are made inconsistent.

In CCS events are labelled by $a, b, \ldots$ or their complements $\bar{a}, \bar{b}, \ldots$ or by the label τ. Let Λ denote this set of labels and $L = \Lambda \cup \{\omega\}$. If x ranges over L the synchronisation algebra for CCS is given by:

$$x \bullet \omega = x$$

$$a \bullet b = \begin{cases} \tau, & \text{if } b = \bar{a} \\ \omega, & \text{if } b \neq \bar{a} \end{cases}$$

The first clause needs a little explanation. If the ω is already the label of an inconsistent event (e.g. a forbidden synchronisation within a component), it may seem puzzling that $x \bullet \omega = x$ rather then $x \bullet \omega = \omega$. However in this case the event labelled $x \bullet \omega$ is made inconsistent by the definition of product, and thus its label does not really matter. We have now the following:

Definition 3.2 *(CCS product) Let $S_0 = (E_0, \# _0, \prec_0, l_0)$ and $S_1 = (E_1, \# _1, \prec_1, l_1)$ be labelled flow event structures. Their* parallel composition *in CCS, noted $(S_0 \parallel S_1)$, is the labelled event structure $(S_0 \times S_1) \upharpoonright (L - \{\omega\})$.*

It should be clear that the parallel composition operator $\parallel$ preserves axiom Δ, since it is defined in terms of general product and restriction. Also, one may easily convince oneself that property Δ is preserved by CCS operators like prefixing and nondeterministic sum (we refer to [BC88] for definition). Hence the flow event structure semantics for CCS is compatible with a categorical one.

4 Conclusions

It is worth noting that the triangle configuration which makes the product fail to be categorical is a rather particular one, where the conflict # does not coincide with the *semantical conflict* $\#_{\mathcal{F}}$ given by: $e \, \#_{\mathcal{F}} e' \iff \nexists$ configuration X s.t. $\{e, e'\} \subseteq X$. This problem obviously does not arise with prime event structures. Structures where $\# = \#_{\mathcal{F}}$ are called *faithful* in [Bou90]. To be sure, one way to avoid the problem with Δ would be to restrict attention to faithful event structures. Any flow event structure may be transformed – or normalised – into a faithful one, having equal events and configurations. However restriction typically creates non faithful structures, as it may be seen from the simple example $(a.b)\backslash a$:

Here the restriction introduces a self-conflict $a \, \# \, a$ while the semantical conflicts $a \, \# \, b$ and $b \, \# \, b$ are left implicit.

We remark that a function space construction can be introduced on flow event structures, so as to give a representation of stable functions on the domains determined by these structures. One can find an example which shows that function space does not preserve property Δ. This is not suggesting a weakness of Δ, though, because the consideration of the stable function space leads to a very different category–the morphisms are different. However, with respect to the category considered in this paper, it is not known if Δ is also necessary for obtaining a categorical product.

Acknowledgements

We would like to thank Gérard Boudol, Mogens Nielsen and Glynn Winskel for their interest and suggestions.

References

[BC88] G. Boudol and I. Castellani. Permutation of transitions: an event structure semantics for CCS and SCCS. In *Proceedings of Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, number 354 in Lecture Notes in Computer Science, pages 411–427, 1988.

[Bou90] G. Boudol. Flow event structures and flow nets. In *Proceedings of LITP Spring School on Semantics of Systems of Concurrent Processes,* La Roche-Posay, number 469 in Lecture Notes in Computer Science, pages 62–95, 1990.

[DDNM88] P. Degano, R. De Nicola, and U. Montanari. On the consistency of "truly concurrent" operational and denotational semantics. In *Proceedings* 3^{th} *Annual Symposium on Logic in Computer Science,* Edinburgh, pages 133–141. IEEE Computer Society Press, 1988.

[GL91] U. Goltz and R. Loogen. Modelling nondeterministic concurrent processes with event structures. *Fundamenta Informaticae*, 14(1):39–73, 1991.

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

[Vaa89] F.W. Vaandrager. A simple definition for parallel composition of prime event structures. Report CS-R8903, CWI, Amsterdam, 1989.

[Win80] G. Winskel. *Events in Computation.* PhD thesis, Department of Computer Science, University of Edinburgh, 1980.

[Win82] G. Winskel. Event structure semantics for CCS and related languages. In M. Nielsen and E.M. Schmidt, editors, *Proceedings* 9^{th} *ICALP,* Aarhus, volume 140 of *Lecture Notes in Computer Science*, pages 561–576, 1982. See also DAIMI Report PB-159, Computer Science Department, Aarhus University, 1983.

[Win87] G. Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency,* *Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course,* Bad Honnef, September 1986, volume 255 of *Lecture Notes in Computer Science*, pages 325–392, 1987.